目　录

第三部分 演化程序

引　言

近 30 年来，人们对借助进化和遗传规律来解决问题的系统产生了越来越浓厚的兴趣：这样的系统维持潜在解的群体，运行一些基于个体适应值的“遗传”算子的选择过程。演化策略(Evolution Strategies)属于其中之一，它是模拟自然进化规律解决参数优化问题的一类算法[319,348]。Fogel 的演化规划(Evolutionary Programming)是一类通过小的有限状态机空间进行搜索的技术[126]。Glover 的分散搜索(Scatter Search)技术是维持一参考点的群体，并通过加权线性组合来产生后代[142]。另一类基于进化的系统就是 Holland 的遗传算法(Genetic Algorithms —— GAs)系统[188]。1990 年，Koza 建议了一个演化系统——遗传规划(Genetic Programming)来搜索解决特定问题的最适计算机程序[231]。

本文使用一个公共的术语：**演化程序**(Evolution Programs —— EP)来表示所有借助进化思想的系统（包括上述的系统）。演化程序的结构如图 0.1 所示：

```
procedure evolution program
begin
    t ← 0
    initialize P(t)
    evaluate P(t)
    while ( not termination-condition) do
    begin
        t ← t+1
        select P(t) from P(t-1)
        alter P(t)
        evaluate P(t)
    end
end
```

图 0.1　演化程序的结构

演化程序是一种概率算法，它维持由许多个体组成的一个群体。在第 t 代，$P(t)=\{x_1^t,\ldots,x_n^t\}$。每个个体表达问题的一个潜在解，任何演化程序是以一些可能很复杂的数据编码 S 来实现的。对每个解 x_i^t 进行评价以给出对其“适应值”的度量。然后，通过选择较适个体的步骤（选择步骤）构成一个新群体（第 $t+1$ 次迭代），一些新群体的成员通过“遗传”算子的转换构成新的解（变换步骤）。其中的一元变异型转换 m_i 通过单个个体 (m_i: $S \to S$) 较小的变换产生新个体，高阶杂交型转换 c_j 通过组合两个或多个个体 (c_j: $S\times\ldots\times S \to S$) 的片断产生新个体。经过若干代后程序收敛。此时奇希望的是最好个体表达一个近似最优的合理解。

考虑下面的例子。假定搜索满足某种要求的一个图，如根据一些准则，搜索一个通讯网络的最优拓扑结构、发送消息的费用、可靠度等。演化程序中的每个个体表达问题的一个潜在解，即每个个体表示一个图。随机产生或是用某种启发式方法产生的图 $P(0)$ 的初始群体为演化程序的初始点（$t=0$）。包括问题要求的评价函数通常是给定的，它返回每个图的适应值、最好个体和最差个体之间的差别。几个变异算子可以用来转变一单个图。几种杂交算子可以把两个或者更多个图的结构组合成一个图。通常，这样的算子包含着与问题有关的知识。例如，如果搜索的图是连通的而且是无环的，即树，一种可能的变异算子可从图中删除一条边，并加入一条新的边以连接两个分开的子图。其他特性是设计与问题有关的变异，在评价函数里并入要求、惩罚非树图等等。

很明显，对一给定问题可以使用多种用公式化表达的演化程序。这些程序在诸多方面是不同的，如可以使用不同的编码结构、转变个体的“遗传”算子、产生初始群体的方法、处理约束的方法和参数（如群体规模、应用不同算子的概率）等。但是，它们都遵循一个共同的规律：由个体组成的群体要经历一些转变，在进化过程中，个体为生存而竞争。

如前所述，演化程序的思想至少已经出现 30 年了[126,188,348]。此后出现了许多不同的演化系统；但本书将从它们的相似性方面讨论演化程序的各种范例。总之，它们之间的主要不同存在于较低的水平上。本书将不讨论各种进化技术之间原理上的不同，即它们是操作于基因型水平还是显型水平。但将从构造一个对特定问题的演化程序方面对它们进行讨论。因此，我们主张对染色体表达使用适当的可能是复杂的数据编码结构以及相应的遗传算子扩展集，而经典的遗传算法使用定长度二进制串作为一个染色体的数据编码 S 来表示个体，并使用两个算子：二进制变异和二进制杂交。换句话说，遗传算法的结构和演化程序（图 0.1）的结构相同，而它们的差别隐藏在较低的水平上。在演化程序中，染色体不必用位串表达，变换的过程包括对给定编码结构和适合于给定问题的“遗传”算子进行变换。

这不完全是一个新的方向。1985 年 De Jong 在[84]中写道：

> “当搜索空间里的元素是由更复杂的数据结构组成的自然表达，如数组、树、有向图等，我们应该做什么？是否有人尝试将它们‘线性化’进一个串表达里，或者是否有方式来创造性地重新定义杂交和变异，使它们能直接操作于这样的结构。我还没有看到这个领域里这方面的任何进展。”

如前所述，遗传算法使用定长度二进制串，且只有两个基本的遗传算子。遗传算法两个主要的早期出版物[188,82]描述了这种理论和实现。正如[155]中所说的：

> “这一工作[82]的贡献是它绝对的抽象化和简化；De Jong 所获得的成功正是由简化得来的。……Holland 的书[188]为 De Jong 的做法提供了理论基础，而且所有随后的遗传算法工作在数学上被认为是相似子集（方案）、最小算子分裂和繁殖选择遗传搜索的组合。……后来的研究者趋向于照搬[188]中的理论建议，因此增强了 De Jong 简洁编码和算子实现上的成功。”

然而，Goldberg 接着说[155]：

“如果不是具有讽刺意义的话，没有人愿意按文献方法原封不动地进行他们的工作。虽然De Jong的实现建立了可以使用的服从Holland理论的简化技术，但随后的研究者趋向于将他们的成就看成是不可亵渎的福音。”

看来对给定问题，一个潜在解的“自然”表达加上相应的“遗传”算子对获得该问题的近似解是有帮助的，而且这种自然建模的方法（演化规划）对一般的问题解决是一个有希望的方向。不同于演化计算的其他范例（如演化策略、演化规划、遗传编程）。遗传算法界的一些研究者开发了其他表达，如次序表（装箱问题）、嵌入表（工厂计划问题）、变元表（半导体线路设计问题）。在过去十年里，提出了各种与应用有关的遗传算法的变种[73,167,171,173,278,363,364,392]。这些变种包括可变长串（串中的元素是if–then–else规则[363]）、比二进制串丰富的编码（如矩阵[392]）和对各种遗传算子修正，它们都是为顺应特定应用的需要而设计的[270]。[285]中将神经网络训练技术的反传法（backpropagation）作为一个算子，并和变异和杂交一起描述了一种适用于神经网络领域的遗传算法。Davis和Coombs描述了用来在设计分组交换通讯网络中的一个阶段所执行的遗传算法[65,76]：使用的表达不是二进制的，且使用了五种分别基于知识的、统计的、数值的“遗传”算子。这些算子和二进制变异和杂交不同。一些研究者在商店调度问题的研究中写道[21]：

“为增强算法的性能并扩展搜索空间，设计了一种储存了与问题有关信息的染色体表达，开发了利用与问题有关的其他信息的重组算子。”

还有很多类似的引证。看来多数研究者是通过使用非串染色体表达或者通过设计适用于特定问题的与问题有关的遗传算子来修正遗传算法的实现。在[228]中，Koza评述道：

“表达是遗传算法的主要问题，因为表达方案严重地限制了系统观察世界的窗口。但是，正如Davis和Steenstrup[74]所指出的，‘在所有Holland及其学生的工作中，染色体都是位串的’，基于串的表达方案是困难的，而且对许多问题是不自然的，有时需要发现作用更强大的表达[84,85,86]。”

各种非标准实现都是针对特定问题而言的。简言之，经典的遗传算法难以直接用到一个问题上，而需要对染色体编码结构做一些修正。本书有意识地和操作于位串的经典遗传算法进行脱离：搜索更丰富的数据编码和用于这些编码上的各种与问题有关的“遗传”算子。通过实算这些结果和算子，我们获得的不再是遗传算法，至少不再是经典的遗传算法的系统。有几节的标题就是“一个修正的遗传算法……”[271]、“特殊化的遗传算法……”[201]、“一个非标准的遗传算法……”[278]。同时，我们感觉到“遗传算法”这个名字可能是对所开发系统的一个误导。Davis开发了几个包含与问题有关算子的非标准的系统。他在[77]中评述道：

“我已看到一些遗传算法界的研究者在摇头……他们坦率地怀疑我们建立的系统是不是遗传算法，因为我们没有使用二进制表达、二进制杂交和二进制变异。”

另外，演化策略是不是一种遗传算法？可否相对地这样说？为避免和演化系统分类

有关的所有问题，我们简单地称它们为“演化程序”(EP)。

为什么要脱离遗传算法而朝向更灵活的演化程序呢？因为尽管有很好的理论，遗传算法在许多领域不能获得成功的应用。造成这种失败的主要原因和它们成功的原因看来是相同的：都是由于域的独立性。

按照域独立性的本义，遗传算法简洁的后果是不能处理非常规约束。如前所述，在大多数遗传算法工作中，染色体是由 0 和 1 组成的位串。在设计解的染色体表达时，要考虑的一个重要问题是与问题有关的约束解的实现。正如[74]中所描述的：

> “不可违反的约束可以通过强加一个较大的惩罚到违反个体上，通过强加中等惩罚或者通过制造不违反约束的个体的表达解码器来实现。如果加一个较高的惩罚到一个违约个体上是可能的，那么我们就会冒着让遗传算法将多数时间花在评价非法个体的风险上。更进一步，当碰巧找到一个合法个体时，它将驱走其他个体并使群体在没有发现更好个体时就收敛，因为一个可能的途径是：其他合法个体的产生需要以非法个体作为中间结构，而对违反约束的惩罚使这样的中间结构的产生成为不可能。如果强加中等的惩罚，同样可能对违反约束个体进行进化，并将它们的分数打得比不违约个体的分数还高，因为是通过接受中度惩罚而不是放弃，有可能使评价函数的其他部分得到更好的演化。如果在评价过程中建立一个解码器来智能地避免构造非法个体，通常的结果是耗费机时的运算。更进一步，这种方法不是让所有的约束都能容易地实现。”

有关解码器和修补算法及几个惩罚函数的方法在 4.5 节给出，其中考虑的是背包问题。

在演化规划里，选择带有一些惩罚的评价函数不是问题，但是选择满足所有问题约束解的“最好”染色体表达及有意义的遗传算子却很成问题。任何遗传算子都将一些特征编码结构从父体传给子代，所以表达编码结构在定义遗传算子中起着重要的作用。而且不同的表达编码结构有不同适合约束的特征，这使问题更加复杂。表达和算子这两个因素是相互影响的；看来任何问题都需要仔细分析才能得到适当的表达及相应的有意义的遗传算子。

Glover 在解复杂的键盘配置问题的研究中[114]写道：

> “虽然遗传搜索方法强有力的特征非常适合键盘构造问题，但位串表达和理想的算子并不能匹配……所要求的约束。例如，如果用三位来表示一个有 40 个键的简单键盘中的每个键，很容易看出任意选择 120 位的编码结构中，每 10^{16} 个才有一个表示一个合法的配置图结构。”

另一个引证可以从 De Jong 的工作中获得[88]，其中对货郎担问题进行了简要的讨论：

> “使用标准的杂交和变异，遗传算法能开发所有城市名的组合空间，但实际上它是一个有趣的排列空间。观察到的问题是当旅途中的城市数 N 增加，排列空间是组合空间越来越小的子集，对拙劣的表达选择强有力的 GA 样本启发式规则就显得越发重要了”

在人工智能(AI)研究的早期，通用问题解决器(General Problem Solvers, GPS)被设计成解决复杂问题的标准工具。但是实际结果是，缘于这些的系统复杂得无法管理，它必须包括

与问题有关的知识。现在，历史又一次重演：直到最近，遗传算法仍被当成用于许多较难的优化问题的标准工具。但是，把与问题有关的知识包括到遗传算法中的需要在一些研究文献中早已明确[10,128,131,170,370]。看来遗传算法（正如 GPS）太具有域独立性而不适合许多应用。所以，在染色体数据编码结构和特定的“遗传”算子中，包括了与问题有关知识的演化程序执行得更好就不奇怪了。

经典的遗传算法和演化程序之间在基本概念上的不同如图 0.2 和图 0.3 所示。经典的遗传算法操作于二进制串，要求将原始的问题修正成适当的遗传算法的形式，这包括潜在解和二进制表达之间的映射、需要很精细的解码器或者修补算法等。这通常不是一个轻松的任务。

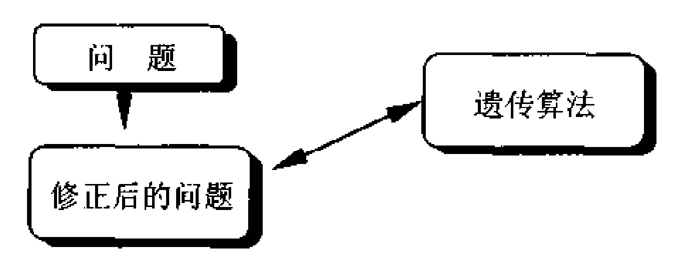


图 0.2 遗传算法

而演化程序不改变问题，而是用“自然的”数据编码结构去改进潜在解的染色体表达，并使用适当的“遗传”算子。

换句话说，为使用演化程序解决一个非常规问题，可以将问题转化为适合遗传算法的形式（图 0.2），也可以将遗传算法转化成适合问题本身（图 0.3）。很明显，经典的遗传算法采用前者，而演化程序采用后者。所以演化程序的思想就像下面的谚语一样简单：

“如果山不能到穆罕默德那里去，那么穆罕默德就到山上去。”

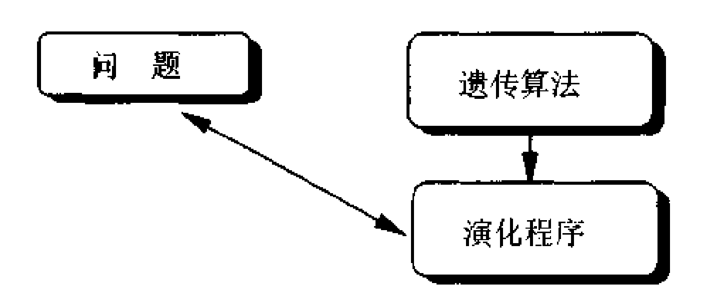


图 0.3 演化程序

这并不是很新的思想。在[77]中，Davis 写到：

“在我看来有时确实是如此，我们不能用二进制表达和只由二进制杂交和二进制变异组成的算子集来处理多数真实世界的问题。一个理由是几乎每个真实世界的域都是与域的知识相关的，这些知识只有当人们考虑将域里的解进行变换时才有可能用得上……，我相信遗传算法是适合于许多真实世界应用问题的算法。我同样相信在人们的算法中应该引入包含这些知识的解码器或扩展算子集。”

我们称这种修正后的遗传算法为“演化程序”。

很难在遗传算法和演化程序之间划一条线。把一个演化程序称为遗传算法要符合什么要求呢？是维持潜在解的群体？是潜在解的二进制表达？还是基于个体适应值的选择过程？或者是重组算子？是模式定理的存在？还是基因块假设？还是上述所有这些因素都要满足？带有整数向量表达和 PMX 算子（第 10 章）的处理货郎担问题的演化程序是遗传算法吗？带有矩阵表达和算术杂交算子（第 9 章）的处理运输问题的演化程序是遗传算法吗？本书不给出上述问题的答案，而只给出一些对各种使用演化规划问题的有趣结果。

如前所述，几位研究者认识到了各种修正的潜力。在[78]中 Davis 写到：

> “当和用户交谈时，我解释我的计划是将遗传算法技术和当前的算法用下面的三个原则进行杂化：
>
> • 使用当前编码。在杂化算法中使用当前算法的编码技术。
>
> • 尽可能地杂化。在杂化算法中合并当前算法中的正面特征。
>
> • 适应遗传算子。用类似于位串杂交和变异算子的方法产生对应于新型编码的杂交和变异。同时合并基于域的启发式算子。
>
> ……我使用了‘杂化遗传算法’一词表示应用这三个原则产生的算法。”

看来，杂化遗传算法和演化程序有共同的思想：脱离经典的位串遗传算法，而朝向更复杂的系统，包括适当的数据编码结构（使用当前的编码）和适当的遗传算子（适应遗传算子）。另外，Davis 假定已经有一种或多种可用于问题域的传统算法，基于这样的算法来讨论所构造的杂化遗传算法。本书的演化程序方法没有做任何类似的假定，后面所讨论的所有演化系统都是随意构造的。

演化程序的强项和弱项是什么？看来演化程序技术的主要强项是有广泛的应用性。本书试图描述各种各样的问题，并讨论对应于每个问题的演化程序的构造。结果常常是杰出的：系统比现有的商业软件执行得要好得多。演化程序的另一个强项是它们的自然并行性。正如[154]中所说的：

> “世界上的序列算法通常并行地穿过无数的陷井和弯路，一个不小的讽刺是遗传算法（高度并行算法）所做的也同样是序列地穿过不自然的陷井和弯路。”

当然，对任何基于群体的演化程序都是这样。另外，我们不得不承认演化程序薄弱的理论基础。对不同的数据编码和修正的杂交和变异要求进行仔细的实算分析，它可以保证合理的执行结果。但这还远远没有完成。

通常，人工智能解决问题的策略可以以“强”和“弱”的方法分类。弱的方法几乎不做有关域的假设；因此它通常有广泛的应用性。而当遇到较大的问题时，它可能会碰到组合爆炸的解题开销[90]。这可以通过对问题域做较强的假设，并随时在解决问题的方法里利用这些假设来避免。但是这种强方法的缺点是它们的应用受限制，甚至于对相关的问题，也经常要求较大程度的重新设计。

演化程序多少处于弱和强方法之间。一些没有做任何有关问题域假设的演化程序是十分弱的，如遗传算法。而另一些带有较大的问题相关性的程序又具有问题的特殊性，

如 GENOCOP 或 GENETIC - 2。GENOCOP（第 7 章）像所有的演化策略（第 8 章）一样是用来解决参数优化问题的。该系统能处理带有任何线性约束集的任何目标函数。GENETIC - 2（第 9 章）适用于解运输问题。其他系统（见第 10 和 11 章）适合于组合优化问题，如日程表问题、货郎担问题、图问题。第 12 章将讨论演化程序在决定规则问题的归纳学习上有趣的应用。

有一点讽刺意义的是：遗传算法经常被看成弱方法；但是，当存在非常规约束时，它们又迅速地变成强方法。不论考虑罚函数、解码器，还是一个修补算法，它们都必须适应特定的应用。而对某些问题，被看成更强、更与问题有关的演化程序又会突然变得弱起来，这将在第 14 章进行更深入的讨论。这表明演化规划方法具有巨大的潜力。

所有这些观察触发了本书对基于复杂的编码结构（不是位串）的不同遗传算子性质的研究。更进一步，这种研究将导致新的编程方法论的产生（见[277]，这种建议的编程方法被称为演化程序设计 EVA —— EVolution progrAmming）[①]。粗略地讲，在这样一个环境下编程的程序员将选择数据编码结构及适合于给定问题的遗传算子，同时还要选择评价函数和初始化群体（其他参数由另一个遗传过程调节）。

但是，在建议构造这样的编程环境之前，还需要许多探索。本书只是通过研究来建立处理不同问题的演化程序编码结构和遗传算子，并朝这一目标迈出第一步。

① 我们将在此书的结尾返回到新的编程环境思想，即第 14 章。



第一部分　遗传算法



第 1 章　遗传算法的主要特征

有一大类有趣的问题，对它们还没有快速合理的算法。其中许多问题都是在应用中经常碰到的优化问题。对一个给定的比较困难的优化问题常能找到有效的算法得到近似最优解。那么我们同样可以采用一些随机算法来处理它。这些算法虽然不能保证获得最优解，但随机地选择充分多个“验证”后，错误概率可能会降低到我们满意的地步。

对许多实际的优化问题，已经有了一些高质量的算法[73]。例如我们可以用模拟退火来解决 VLSI 设计中布线方案及组件放置问题，或者是货郎担问题。另外，许多大型的可能是 NP 难解的组合优化问题都能在现代的计算机上用 Monte Carlo 技术获得近似解。

一般来说，任何难解的任务都可被看成是要解决一个问题，反过来，它也可以被看成是对潜在解空间的一种搜索。因为我们最终的目的是要获得“最好”解，所以可以把这种任务看成是一个优化过程。对小空间，经典的穷举法就足够用；而对大空间，则需要使用特殊的人工智能技术。遗传算法就是这些技术中的一种；它们是一类模拟某些自然现象的随机搜索算法，是模拟遗传继承和达尔文的适者生存原理。正如参考文献[74]中所说：

> “……遗传算法的思想和自然进化的思想是同出一辙的。在进化过程中，每个种群所面临的问题是寻找一种对复杂和变化着的环境最有利的适应方式。每个种群所获得的“知识”都被嵌入其成员的染色体组成当中。”

遗传算法的思想正是做自然界想做的事情。让我们以兔子作为例子：在一给定时间里，有一群兔子。其中一些比另外一些兔子跑得快，而且更聪明。这些兔子被狐狸吃掉的可能性比较小。因此它们中的多数就存活下来并繁殖更多的兔子。当然，一些慢而愚笨的兔子也会存活下来，只是因为它们比较幸运。这些存活的兔子群体开始生育。生育的结果是兔子遗传材质的充分融合：一些慢的兔子生出快的兔子，一些快的生出更快的，一些聪明的兔子生出愚笨的兔子，等等。在最顶层，自然界时不时地变异一些兔子的基因材质。所产生的小兔子平均来说要比原始的群体更快更聪明，因为从狐狸口中生存下来的父代多数是跑得更快也更聪明的兔子。同样，狐狸也经历相似的过程——否则兔子可能跑得太快又太聪明以致狐狸根本抓不到了。

遗传算法遵循着和兔子的故事极相似的过程。在对遗传算法的结构进行深入了解之前，我们首先快速回顾一下遗传学的历史（出自[380]）：

> “作为进化规律的自然选择早在遗传机制被发现以前就由 C. Darwin 提出来了。在尚不知道基本的遗传规律时，达尔文假定了遗传融合或混合，即假设父母的品质就像流体一样在后代中混合在一起。他的自然选择理论遭到过强烈的反对，首先 F. Jenkins 认为快速的交叉会很快消除任何遗传的差别，在同类群

体中也就不存在选择性，即所谓的‘Jenkins 之梦’（Jenkins nightmare）。

直到 1865 年，G. Mendel 发现了遗传因子从父代到子代转移的基本原理，它展示了这些因子的不同性质，‘Jenkins 之梦’才得到解释，因为有了这种差异，遗传的区别才不会‘消融’。

Mendel 法则自它 1900 年被 H. de Vries、K. Correns 和 K. von Tschermak 各自独立地发现以来，就闻名于科学界。遗传学是由 T. Morgan 及其合作者完整发展起来的，他们用实验证明，染色体是转移遗传信息的载体，代表着遗传因子的基因在染色体上是线性排列的。随后，逐渐积累的实验事实显示，Mendel 法则对所有的有性繁殖生物体都是适用的。

但是，尽管 Mendel 法则和达尔文的自然选择理论仍保持为独立的、互不相连的概念。而且它们彼此相对。直到 1920 年（见 Cetverikov[61]所做的经典工作），人们证明 Mendel 的遗传学和 Darwin 的自然选择理论不是冲突的，它们之间的相互结合构成了现代的进化理论。”

遗传算法借用遗传学的词汇。我们将讨论一个群体中的个体（或基因型、结构）：这些个体常常又被称为串或者是染色体。这可能产生一些误导：对一给定种群，每个器官的每个细胞都携带一定数量的染色体（例如，人为 46 个），但在本书中，我们只讨论一个染色体的个体，即单倍体(haploidy)，有关二倍体(diploidy) —— 一对染色体控制信息的情况及其在遗传算法中的相关概念，读者可以参考[154]及最近由 Greene [165]、Ng 和 Wong [298]所做的工作。染色体的基本单位是基因（同时也是特征、性质或解码）线性安排的序列。每个基因控制一个或者几个遗传特征。某一特征的基因位于染色体特定的位置，被称为位（locus，串位置）。个体的任何特征，如头发的颜色，表征其自身的不同。在几种状态下的基因被称为等位基因(alleles)，即特性值。

每个基因型（本书中为一单个染色体）代表一个问题的在特定染色体意义上的潜在解，即其显形（表现型）是由用户外部定义的。一个发生在染色体群上的进化对应于一个潜在解空间的搜索。这样的搜索要求两个明显冲突方面之间的平衡：开发(Exploitation)最好解及探究(Exploration)搜索空间[46]。爬山法(Hillclimbing)是尽可能地改进开发最好解策略的例子，但它忽视了对搜索空间的探究。随机搜索(Random search)是一个探究搜索空间而忽视开发希望空间的典型例子。遗传算法则是在总体上寻求显著的开发和空间探究之间平衡的一类算法。

遗传算法在用于优化问题上确实是成功的，像布线方案(wire routing)、调度问题(scheduling)、自适应控制(adaptive control)、游戏规则(game playing)、认知模型(cognitive modeling)、运输问题(transportation problems)、货郎担问题(traveling salesman problems)、优化控制(optimal control)问题、数据库查询优化(database query optimization)等等。（见[15]、[34]、[45]、[84]、[154]、[167]、[170]、[171]、[273]、[344]、[129]、[103]、[391]、[392]）但是，De Jong [84]并不认为遗传算法是一个优化工具：

“……因为以上的研究主要集中于函数优化的应用，所以很容易认为遗传算法本身是优化算法，而当遗传算法不能找到特定搜索空间的明显最优值时，就会

变得很惊讶或失望。我的建议是为避免这种认识上的错误，把遗传算法看成是一个对自然过程高度理想化的模拟，是对自然过程的目标和意图（如果有的话）的具体化。我不敢确信是否有人正在承担着定义进化系统的目标和意图的任务；但是，我敢说这样的系统不能在总体上被当作是函数优化器”。

另一方面，优化是遗传算法应用的主要领域。Schwefel 说[348]：

“很少有一本现代的杂志，不论是工程、经济、管理、数学、物理或者是社会科学杂志。其中没有‘优化’这个关键词。如果概括所有专家的观点，问题可概述为从许多可能事件状态中选择一个较好或最好的（按照 Leibniz 的优化说法）。”

在过去十年，优化的重要性不断提高，许多重要的大规模组合优化问题及高度约束的工程问题只能用现代的计算机获得近似解。

遗传算法所要解决的就是这类复杂的问题。遗传算法属于概率算法一类，尽管使用了直接、随机搜索的方法，但遗传算法和随机算法却是不同的。遗传算法比现在的直接搜索方法更强大。基于搜索的这种算法另一个重要性质是它们维持一个潜在解的群体，而其他所有的方法都是处理搜索空间中的单个点。

爬山法使用了迭代改进的技术，该技术应用于搜索空间的单个点，即当前点。在迭代过程中，一个新点是从当前点附近的点中选出的，这就是它又被称为邻近搜索或局部搜索的原因[233]。如果新点的目标函数值更好①，那么该新点就变成当前点。否则就选择和测试其他当前点的邻近点。如果没有更进一步可能的改进，则算法终止。

很明显，爬山法只能提供局部最优值，这些值依赖于初始点的选择，且无法知道找到的解相对于全局最优解的相对误差。为增加成功的机会，爬山法通常从不同的开始点执行多次，这些点不必是随机选择的——开始点的选择可依据于先前的执行结果。

模拟退火(Simulated annealing)技术[1]消除了爬山法的许多缺点：解不再依赖初始点，而且通常靠近最优点。这主要是通过引入接受概率 p 完成的（即用一个新点替换当前点）：如果新点的目标函数值更好，则 $p=1$；否则 $p>0$。对后一种情况，接受概率 p 是当前点、新点的目标函数及另一个控制参数“温度” T 的函数。总的来说，低的温度 T 表示新点的接受概率较小。在算法执行过程中，系统的温度 T 随着算法的进行而降低。最后终止于不再有可接受变化发生的低的 T 值。

如前所述，遗传算法通过保持一个潜在解的群体执行了多方向的搜索并支持这些方向上的信息构成和交换。群体经过一个模拟进化的过程：在每一代，相对“好”的解产生，相对“差”的解死亡。为区别不同的解，我们使用了一个目标（评价）函数，它起着一个环境的作用。

本章的第 1.4 节给出了一个有关爬山法、模拟退火法和遗传算法的例子。

简单遗传算法的结构和演化程序的结构是相同的（见引言部分的图 0.1）。在第 t 个迭代，遗传算法维持一个潜在解的群体（染色体、向量），$p(t)=\{x_1^t,\ldots,x_n^t\}$。每个解 x_i^t

① 对求最小值问题为一更小值，对求最大值问题为一个更大值。

用其“适应值”进行评价。然后通过选择更适个体（t+1 此迭代）形成一个新的群体。新群体的成员通过杂交和变异进行变换，以形成新的解。杂交组合了两个亲体染色体的特征，并通过交换父代相应的片断形成了两个相似的后代。例如，如果父代用五维向量(a_1,b_1,c_1,d_1,e_1)和(a_2,b_2,c_2,d_2,e_2)表示，在第二个基因后杂交，染色体将产生后代(a_1,b_1,c_2,d_2,e_2) 和 (a_2,b_2,c_1,d_1,e_1)。杂交算子的意图是在不同潜在解之间进行信息交换。

变异是通过用一个等于变异率的概率随机地改变被选择染色体上的一个或多个基因。变异算子的意图是向群体引入一些额外的变化性。

对一给定问题，遗传算法和任何演化程序一样必须经过下面的五个步骤：

- 对问题潜在解的遗传表达。
- 产生潜在解初始群体的方法。
- 起环境作用的用“适应值”评价解的适应程度的评价函数。
- 改变后代组成的各种遗传算子。
- 遗传算法所使用的各种参数：群体规模、应用遗传算子的概率等。

我们将通过三个例子讨论遗传算法的主要特征。第一个例子用遗传算法优化一个简单的实数变量函数。第二个例子说明遗传算法如何学习一个简单的游戏策略 —— 囚犯困境。第三个例子为遗传算法求解组合 NP 难解问题的一个应用实例 —— 货郎担问题。

1.1 简单函数的优化

本节将讨论用遗传算法优化一个简单单变量函数的基本特征。函数的定义是：

$$f(x) = x\cdot\sin(10\pi\cdot x) + 1.0$$

其图如图 1.1 所示，问题是：找到区间[−1..2]内的 x 使函数 f 值最大，即找出 x_0，使对所有 $x\in[-1..2]$，$f(x_0)\geq f(x)$。

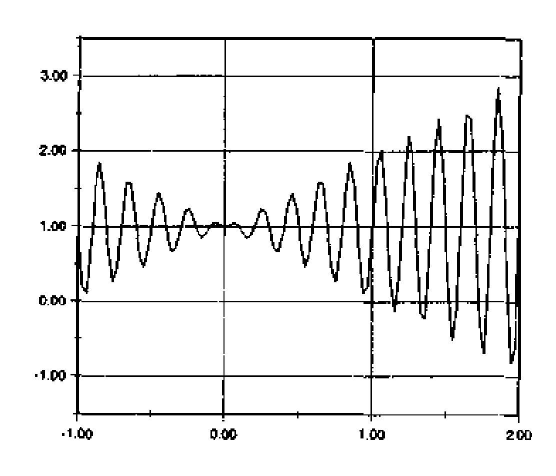


图 1.1 函数 $f(x) = x\cdot\sin(10\pi\cdot x) + 1.0$ 图

很容易进行函数 f 的分析，可以确定该函数的一阶导数 f' 为零：

$$f'(x)=\sin(10\pi\cdot x)+10\pi x\cdot\cos(10\pi\cdot x)=0$$

该式等同于： $\tan(10\pi\cdot x)=-10\pi x$

很明显，上述方程有无穷多解，

$$x_i=\frac{2i-1}{20}+\varepsilon_i,\quad 其中\ i=1,2,\ldots$$

$$x_0=0$$

$$x_i=\frac{2i+1}{20}-\varepsilon_i,\quad 其中\ i=-1,-2,\ldots$$

这里ε_i表示接近于零的实数递减序列（i=1, 2, ..., 及 i= −1, −2, ...）。

注意对x_i，如果i是一个奇数，函数f达到其局部最大。如果i是一个偶数，则达到其最小，见图1.1。

因为问题的定义域是$x\in[-1..2]$，当$x_{19}=\frac{37}{20}+\varepsilon_{19}=1.85+\varepsilon_{19}$时，函数达到其最大。此时$f(x_{19})$比$f(1.85)=1.85\cdot\sin(18\pi+\frac{\pi}{2})+1.0=2.85$稍大。

假定希望构造一个遗传算法来解决上述问题，即求函数f的最大值。下面的部分将依次讨论遗传算法的主要组成。

1.1.1 表 达

这里使用一个二进制向量作为一个染色体来表示变量 x 的真值。向量的长度依赖于要求的精度，本例为小数点后六位数。

变量x的区间长为3，精度要求意味着区间[−1..2]应该至少被划分成3×1000000个等长区间。这就意味着二进制向量（染色体）要求有22位：

$$2097152=2^{21}<3000000<2^{22}=4194304$$

在区间[−1..2]内，从二进制串$\langle b_{21}b_{20}\ldots b_0\rangle$到实数$x$之间的映射是直接的，这可以通过两步完成：

- 将二进制串$\langle b_{21}b_{20}\ldots b_0\rangle$从二进制转换到十进制：

$$(\langle b_{21}b_{20}\ldots b_0\rangle)_2=(\sum_{i=1}^{21}b^i\cdot 2^i)_{10}=x',$$

- 找到相应的实数x，使：

$$x=-1.0+x'\cdot\frac{3}{2^{22}-1}$$

这里−1.0是区间左边界，3是区间长度。

例如，染色体（1000101110110101000111）表示数0.637197，

因为 $x'=(1000101110110101000111)_2=2288967$

$$x=-1.0+2288967\cdot\frac{3}{4194303}=0.637197。$$

当然，染色体（0000000000000000000000）和（1111111111111111111111）分别表示区间的边界 −1.0 和 2.0。

1.1.2 初始群体

初始化过程非常简单：产生一个染色体群体，其中每个染色体都是 22 位的向量。每个染色体的所有 22 位都是随机初始化的。

1.1.3 评价函数

对二进制向量 v 的评价函数 *eval* 等同于函数 f 。

$$eval(v) = f(x)$$

这里染色体 v 表示实数值 x。

正如前面已经注意到的，评价函数起着环境的作用，它们的适应值被用来评价潜在解。例如，三个染色体：

$$v_1 = (1000101110110101000111)$$

$$v_2 = (0000001110000000010000)$$

$$v_3 = (1110000000111111000101)$$

分别对应于值 $x_1 = 0.637197$、$x_2 = -0.958973$ 及 $x_3 = 1.627888$。因此评价函数对它们进行如下评价：

$$eval(v_1) = f(x_1) = 1.586345$$

$$eval(v_2) = f(x_2) = 0.078878$$

$$eval(v_3) = f(x_3) = 2.250650$$

很明显，评价值最高的染色体 v_3 是三个染色体中最好的一个。

1.1.4 遗传算子

在遗传算法的变换阶段，我们将使用两个经典的遗传算子：变异和杂交。

如前所述，变异以等于变异率的概率改变染色体上的一位或多位上的基因。假定选择染色体 v_3 上第 5 位基因来进行变异。因为该染色体上的第 5 位基因是 0，它将变成 1。因此，染色体 v_3 经过变异后将变成

$$v_3' = (1110\ 100000111111000101)$$

该染色体表达值 $x'_3 = 1.721638$ 和 $f(x'_3) = -0.082257$。这意味着变异导致染色体 v_3 值大为减少。如果选择染色体 v_3 上的第 10 位基因来进行变异，那么

$$v_3''=(111000000\ 1111111000101)$$

对应的值为 $v_3'' = 1.630818$ 和 $f(v_3'') = 2.343555$，即原始值 $f(x_3) = 2.250650$ 得到了改进。

下面以染色体 v_2 和 v_3 来说明杂交算子。假定杂交点随机地选择在第 5 位基因后：

$$v_2 = (00000|01110000000010000)$$

$$v_3 = (11100|00000111111000101)$$

产生的两个子代是

$$v_2' = (00000|00000111111000101)$$

$$v_3' = (11100|01110000000010000)$$

这两个子代的评价值为

$f(v_2')=f(-0.998113)=0.940865$
$f(v_3')=f(1.666028)=2.459245$

注意第2个子代，其适应值比其父代都好。

1.1.5 参数

对该问题使用了下面的参数：群体规模 $pop_size=50$、杂交率 $p_c=0.25$ 和变异率 $p_m=0.01$。下面给出该遗传系统的一些实算结果。

1.1.6 实算结果

表1.1给出了评价函数改进明显的代数及相应函数值。经过150代，最好染色体是：

$$v_{max}=(1111001101000100000101)$$

对应的值 $x_{max}=1.850773$。

表1.1 150代的结果

代 数	函数评价值	代 数	函数评价值
1	1.441942	39	2.344251
6	2.250003	40	2.345087
8	2.250283	51	2.738930
9	2.250284	99	2.849246
10	2.250363	137	2.850217
12	2.328077	145	2.850227

正如所预计的，$x_{max}=1.85+\varepsilon$，　$f(x_{max})$ 比2.85稍大。

1.2 囚犯困境

本节将解释遗传算法如何用于学习一个简单的游戏策略，即囚犯困境。这里使用Axelrod[14]获得的结果。

两个囚犯被关在一个隔离的牢房里，彼此不能交流。每个囚犯都分别被要求背叛另一个囚犯或者与另一个囚犯合作。如果只有一个囚犯选择背叛，该囚犯就会被奖励，另一个囚犯被惩罚。如果两个都选择背叛，则两个都被监禁并受到折磨。如果两个都选择合作，则两个都会受到中等的奖励。这样，不管另一个囚犯怎样选择，选择背叛总是比选择合作产生更高的报酬。但是如果两个都选择背叛，其奖励要比两个都选择合作得到的奖励少。囚犯困境是要决定是选择与另一个囚犯合作还是选择背叛。

表1.2 囚犯困境游戏奖分表：p_i 为对第 i 个玩家的奖分

玩家1	玩家2	p_1	p_2	评注
背叛	背叛	1	1	相互背叛受惩罚
背叛	合作	5	0	诱发背叛受到奖励
合作	背叛	0	5	受到奖励诱发背叛
合作	合作	3	3	对相互合作的奖励

这个游戏可以当作两个玩家的游戏来玩，轮到某个玩家时，他要么选择与其他囚犯合作要么选择背叛。然后按照下面的表 1.2 对玩家进行计分。

遗传算法如何被用来学习囚犯困境的策略呢？遗传算法必须保持一个“玩家”的群体，它们中的每一个都表示一个特定的策略。初始阶段，某个“玩家”的策略是随机选择的。此后的每一步是，玩游戏、记录分数。一些玩家被选择产生下一代，一些被选择配对。当两个玩家配对就产生新的玩家，其选择方案的构筑来自于其父代（杂交）。变异通常通过随机改变这些策略的表达式在玩家的策略里引入一些变化。

1.2.1 策略表达

首先需要表达一个策略的某种方法，即一个潜在解。为简化处理，考虑确定性的策略并使用前三步选择当前步。对每一步一共有四种可能的结果，即总共有 $4 \times 4 \times 4 = 64$ 种不同的前三步记录。

这种类型的策略通过指定可能记录中的每一个该如何选择来表达。这样，一个策略可以用一个 64 位的串表达（背叛或合作 —— D 或 C），它表示 64 种可能记录中的每一个该如何选择。为获得游戏开始的策略，需要对游戏开始之前的三个假想的行为进行初步假设。这就要求六个额外的基因，即染色体共有 70 位。

这 70 位的串指明玩家在每种可能的环境下将如何作，因此也就完全定义了一个特定的策略。这个拥有 70 个基因的串作为玩家的染色体用于演化过程。

1.2.2 遗传算法的轮廓

Axelrod 给出的学习囚犯困境策略的遗传算法有以下四步：

（1）选择一初始群体。每个玩家被随机地分配一个 70 位的串，以代表上述的一个策略。

（2）测试每个玩家以确定其效力。每个玩家使用其染色体定义的策略和另一个玩家玩游戏。玩家的分数取整个游戏分数的平均。

（3）选择玩家繁殖。一个具有平均分数的玩家给定一次配对；高于平均一个标准偏差的玩家给定两次的配对；低于平均一个标准偏差的玩家不给配对。

（4）成功的玩家随机地配对，每次配对产生两个后代。每个后代的策略是由其父代的策略确定的。这是通过使用两个遗传算子获得的：杂交和变异。

经过这四步，就获得一个新群体。新群体将更趋向于先前代中成功个体的行为模式，而不趋向于不成功个体的行为模式。对新一代，相对高分数的个体将有较大的可能将它们的策略片断传递下去，而相对不成功个体，将有较少的可能将其策略的片断传递下去。

1.2.3 实算结果

运行该程序，Axelrod 获得了十分明显的结果。从一个严格的随机点开始，对应遗传算法演化群体中中等个体对应的函数值就已和很知名的启发算法得到的结果相同。其运

行行为模式包含在大多数的个体中，它们是：

(1) 稳定合作模式：经过 3 次连续合作后继续合作。①

[即，　(CC)(CC)(CC)后仍是 C]

(2) 惩罚背叛模式：当其他玩家选择背叛时背叛。

[即，　(CC)(CC)(CD)之后为 D]

(3) 不记前嫌模式：当合作恢复后继续合作。

[即，　(CD)(DC)(CC)后为 C]

(4) 巩固合作模式：合作恢复后合作。

[即，　(DC)(CC)(CC)后为 C]

(5) 接受常规模式：三次互相背叛后背叛。

[即，(DD)(DD) (DD)后为 D]

详情见[14]。②

1.3　货郎担问题

这一节解释遗传算法怎样用于货郎担问题（TSP）上。这里只讨论一种方法，TSP 的其他方法将在第 10 章中讨论。

简单地说，货郎担问题是：货郎必须访问其版图上的所有城市，每次一个，最终回到开始点；给定所有城市之间的旅费，他应该如何计划其路线以获得最小的全程开销？

TSP 是出现在许多应用中的一个组合优化问题。已有几种解决该问题的分支定界法、近似算法及启发搜索算法。近些年，也有一些遗传算法的尝试获得了 TSP 问题的近似解[154]，这里我们只讨论其中之一。

首先讨论一个与染色体表达有关的问题：应该对染色体使用整数向量，还是将它们转换成二进制串？在先前的函数优化和囚犯困境例子中，我们对染色体或多或少自然地使用了二进制向量。这就允许二进制变异和杂交的使用。应用这些算子，可以获得合法的后代，即得到搜索空间里的后代。而对货郎担问题却不是这样。在一个有 n 个城市的 TSP 问题的二进制表达里，每个城市都应该被编码成有 $\lceil \log_2 n \rceil$个位的串，染色体是有 $n \cdot \lceil \log_2 n \rceil$个位的串。变异可能产生一个非法的城市序列，即可能在一个序列中两次到达同一城市。而且，对一个有 20 个城市的 TSP（这里对每个城市需要 5 位），一个 5 位的序列（如 10101）并不对应于任何城市。当使用杂交算子时，同样存在这样的情况。很明显，如果使用先前定义的变异和杂交，需要一些“修补算法”。这样的算法将“修补”一个染色体，将其返回到搜索空间里。

看起来整数向量表达更好些，不必使用修补算法，可以将与有关问题的知识加入

---

① 最好三次行动由$(a_1b_1)(a_2b_2)(a_3b_3)$描述，这里 a 是一个玩家的行动（C 为合作，D 为背叛），b 是另一个玩家的行为。

② 囚犯困境问题可以被推广到不止两个玩家：细节和实验结果见 Yao 和 Darwen[413]的工作。第 13 章和[120]都有用演化规划技术解该问题的讨论。

到算子中。这种方式能够"智能地"避免构造非法个体。并可以接受整数表达。 向量 $v=\langle i_1\, i_2 \dots i_n \rangle$ 代表一个旅行：从 i_1 到 $i_2,\dots$，从 i_{n-1} 到 i_n 后返回 i_1。v 为〈1 2 … n〉的排列。

对个体的初始化可以使用一些启发性规则，如，可以接受从不同城市开始的 TSP 贪婪算法的一些结果，或者可以通过一个随机的排列〈1 2 … n〉的样本对群体初始化。

对染色体的评价是直接的：给定所有城市间的旅费，可很容易地算出全程总费用。

TSP 是搜索旅行中的最佳城市次序。使用搜索更好串序的一元算子是很容易的。但是，只使用一元算子，几乎没有希望找到更好的次序，更不必说最好的[160]。而且，遗传算法的强项就是高适应个体杂交组合进行结构信息交换。所以需要一个杂交算子。出于这样的目的，这里使用了一个变种的 OX 算子的[71]，它通过从一个父体中选择旅行序列并保存另一个父体的相对城市次序来构造后代。如果父代为

〈1 2 3 4 5 6 7 8 9 10 11 12〉和〈7 3 1 11 4 12 5 2 10 9 6 8〉

且，被选择的部分为： (4 5 6 7)

则产生的后代为 〈1 11 12 4 5 6 7 2 10 9 8 3〉

正如所要求的，后代构成了两个父体的结构关系。在构筑第二个后代时，则父体角色被颠倒。

基于上述的遗传算法执行了随机搜索，但提供了更多的改进空间。当应用于有 100 个随机城市的 TSP 时，经过 20000 代后，该算法 20 次随机运行平均的典型结果得到了一个高于已知最优值 9.4%的全程值。

对 TSP 的完整讨论、表达和使用的遗传算子，读者可参考第 10 章。

1.4 爬山法、模拟退火法和遗传算法

本节将讨论三个算法，即爬山法、模拟退火法和遗传算法，用于一个简单的优化问题的实例。通过该例子，本文将勾勒出遗传算法的特性。

搜索空间是一长度为 30 的二进制串 v。目标函数 f 为求：

$$f(v)=|11\cdot one(v)-150|$$

的最大，这里函数 $one(v)$返回串 v 中 1 的数目。

例如，与下面三个串

$$v_1=(110110101110101111111011011011)$$
$$v_2=(111000100100110111001010100011)$$
$$v_3=(000010000011001000000010001000)$$

相对应于的是：

$$f(v_1)=|11\cdot 22-150|=92$$
$$f(v_2)=|11\cdot 15-150|=15$$
$$f(v_3)=|11\cdot 6-150|=84$$

因为 $one(v_1)=22$，$one(v_2)=15$ 及 $one(v_3)=6$

函数 f 是线性的，而且优化起来很容易。这里只是用它来说明这三个算法的思想。函数 f 的特征是它有一个全局最优解，即

$$\boldsymbol{v}_g = (111111111111111111111111111111)$$

$$f(\boldsymbol{v}_g) = |11 \cdot 30 - 150| = 180$$

而且有一个局部最大

$$\boldsymbol{v}_l = (000000000000000000000000000000)$$

$$f(\boldsymbol{v}_l) = |11 \cdot 0 - 150| = 150$$

这里有几个爬山法的版本。它们在新串的选择上是不同的。第一个版本是一个简单的迭代爬山算法（*MAX* 次迭代），如图 1.2 所示 （最速上升爬山法）。

```
procedure iterated hillclimber
begin
    t ← 0
    repeat
        local ← FALSE
            select a current string v_c at random
        evaluate v_c
        repeat
            select 30 new strings in the neighborhood of v_c by flipping single bits of v_c
            select the string v_n from the set of new strings with the largest value of
                                objective function f
            if  f(v_c) < f(v_n)
                then  v_c ← v_n
                else  local ← TRUE
            until local
            t ← t + 1
    until  t = MAX
end
```

图 1.2　简单（迭代）爬山法

最开初，所有的 30 个邻居都被考虑，选择 $f(\boldsymbol{v}_n)$ 最大的那个向量 $\boldsymbol{v}_n$ 和当前串 $\boldsymbol{v}_c$ 竞争。如果 $f(\boldsymbol{v}_c) < f(\boldsymbol{v}_n)$，则该串将变成当前串。否则，没有局部改进：算法到达局部或者全局最优（$local \leftarrow$ TRUE）[①] 。算法的下一次迭代（$t \leftarrow t+1$）在新的随机点上执行。

有趣的是，上述爬山算法的一个单个迭代的成败（即返回全局还是局部最优值）是由随机选择的初始串决定的。很明显，如果初始串有 13 个 “1” 或者更少的 “1”，算法将总是终止于局部最优（失败）。理由是有 13 个 “1” 的串返回的目标函数值是 7，任何向全局最优靠近的单步改进，即 “1” 的数目增加到 14，都将使目标函数值减小到 4；而 “1”数目的减小将增大函数值：有 12 个 “1” 的串产生的值为 18，有 11 个 “1” 的串产生的值为 29，等等。这就迫使搜索向一个 “错误的” 方向进行，而停留在局部最大。

对有许多局部最优点的问题，运行一个简单迭代找到全局最优的机会是很渺茫的。

模拟退火的过程如图 1.3 所示。

函数 *random*[0,1)返回的是在区间[0,1)里的一个随机数。（termination–condition）用来检查是否达到 “热平衡”，即是否被选择新串的概率分布接近 Boltzmann 分布[11]。在某

① 译者注：原书印刷有误。

些情况下[4]，这一重复迭代将被执行 k 次，k 是该方法的另一个参数。

```
procedure simulated annealing
begin
    t ← 0
    initialize temperature T
    select a current string v_c at random
    evaluate v_c
    repeat
        repeat
        select a new string v_c in the neighborhood of v_c by flipping a single bit of v_n
        if  f(v_c) < f(v_n)
            then  v_c ← v_n
            else if  random[0,1) < exp((f(v_n) − f(v_c))/T)
                then v_c ← v_n
        until  (termination–condition)
        T ← g(T, t)
        t ← t + 1
    until (stop–criterion)
end
```

图 1.3 模拟退火

在步骤[$g(T,t)<T$]，温度 T 将降低。算法终止于较低的 T 值：（stop–criterion）检查是否系统被冻结，即最终是否不再有可接受的变化。

如前所述，模拟退火算法能够避开局部最优，考虑有 12 个“1”的串：

$$v_4=(111000000100110111001010100000)$$

其评价值 $f(v_4)=|11\cdot12-150|=18$。对作为初始串的 v_4，前面的爬山法只能获得局部最大

$$v_1=(000000000000000000000000000000)$$

因为任何有 13 个“1”的串（即趋向于全局最优的步骤）评价值为 7，小于 18。而模拟退火算法把有 13 个“1”的新串作为当前串，其概率为：

$$p=\exp\{(f(v_n)-f(v_c))/T\}=\exp\{(7-18)/T\}$$

对当前串的某温度，设为 $T=20$， 得

$$p=e^{-\frac{11}{20}}=0.57695$$

即接受的概率大于 50%。

正如在第 1.1 节中所讨论的，遗传算法维持一个串的群体。两个相对差的串

$$v_5=(111110000000110111001110100000)$$

$$v_6=(000000000001101110010101111111)$$

都得到评价值 16。但如果在第 5 位置和第 12 位置之间进行杂交，它们将产生更好的后代：

$$v_7=(111110000001101110010101111111)$$

新的后代 v_7 的评价值为：

$$f(v_7)=|11\cdot19-150|=59$$

有关详细的讨论和其他算法（各种爬山法、遗传搜索和模拟退火的变种）对具有不

同特征的函数的测试，读者可以参考[4]。也可以构造包括几种算法技术的混种 —— 如动态爬山技术[92]。这里给出最近出现在 Internet（comp.ai.neural–nets [337]）的有趣评述：它给出了爬山法、模拟退火和遗传算法形象的比较：

"注意，在目前讨论的所有爬山法中，袋鼠最有希望到达靠近它出发点的山顶。但不能保证该山顶是珠穆朗玛峰，或者是一个非常高的峰。各种使用的方法都试图找到实际全局最优值。

在模拟退火中，袋鼠喝醉了，而且随机地跳跃了很长时间。但是，它渐渐清醒了并朝着峰顶跳去。

在遗传算法中，有很多袋鼠，它们降落到喜玛拉雅山脉的任意地方（如果飞行员没有迷失方向）。这些袋鼠并不知道它们被设想寻找珠穆朗玛峰。但每过几年，你就在一些高度较低的地方射杀一些袋鼠，并希望存活下来的那些是多产的，并在那里生儿育女"。

1.5 结 论

遗传算法对函数优化、囚犯困境和货郎担问题的三个例子展示了遗传算法广泛的应用。同时应该观察到存在着潜在难点。对货郎担问题表达的观点并不十分明显。新使用的 OX 杂交算子是不常用的。对其他有难度的问题，我们是否会遇到有更大困难的算子呢？在第一和第三个例子里（函数优化和货郎担问题），评价函数是显式定义的：在第二个例子里（囚犯困境），用一个简单的模拟过程给出了对染色体的评价，我们测试每个玩家以确定其效力：每个玩家使用其染色体定义的策略来和其他玩家玩游戏，玩家的分数是其游戏的总评价值。我们应该怎样处理非显示定义的评价函数呢？例如布尔量满足问题（SAT）？看起来有自然的串表达（第 i 位表示第 i 个布尔变量的真值）。但是，选择评价函数的过程是很难的[90]。

优化非约束函数的第一个例子允许我们方便地使用表达，其中任何二进制串对应于该问题域里（即[−1, 2]）的一个值。这就意味着变异和杂交都能产生合法的后代。对第二个例子也是这样：任何位的组合代表一个合法的策略。第三个问题有一个约束条件：每个城市在一个合法的旅行中只应该精确地出现一次。这就引发一些问题：我们使用整数向量（而不是二进制表达），而且修正了杂交算子。但是应该怎样总体上处理约束呢？是否存在其他方法呢？

回答这些并不是很容易的，我们将在本书的后面章节讨论这些问题。

第2章　遗传算法的运行步骤

本章讨论用遗传算法解一个简单参数优化问题的过程。从一些一般性的评述开始，随后是详细的例子。

注意，为不失一般性，这里的问题都是求最大值问题。如果优化问题是求函数 f 的最小值，它等同于求函数 g 的最大，其中 $g=-f$，即

$$\min f(x) = \max g(x) = \max\{-f(x)\}$$

假定目标函数 f 在其域里只取正值；若为负，可通过加入某个正常数 C 使之为正，即

$$\max g(x) \to \max\{g(x)+C\}$$ [①]

现在，假设欲求一个有 k 个变量的函数 $f(x_1,\ldots,x_k)$ ：$R^k \to R$ 的 f 的最大值。进一步假设每个变量 x_i 为域 $D_i=[a_i, b_i]\subseteq R$ 内的一个值，且对所有 $x_i\in D_i$，$f(x_1,\ldots,x_k)>0$。假定以某个要求的精度优化函数 f：这里取自变量小数点后第6位。

很明显，要达到这样的精度，每个域 D_i 应该被分割成 $(b_i-a_i)\cdot 10^6$ 个等尺寸的区间。这里用 m_i 表示使 $(b_i-a_i)\cdot 10^6 \le 2^{m_i}-1$ 成立的最小整数。这样，对每个变量 x_i，由串长为 m_i 的二进制编码表达显然能满足精度要求。下面的公式对应于每个串的自变量的值：

$$x_i = a_i + decimal(1001\ldots001_2)\cdot\frac{b_i-a_i}{2^{m_i}-1}$$

其中 $decimal(string_2)$ 表示二进制串的十进制值。

现在，代表一个潜在解的染色体被长度为 $m=\sum_{i=1}^{k} m_i$ 的二进制串表达；前 m_1 位对应区间 $[a_1, b_1]$ 里的一个值，随后的 m_2 位对应区间 $[a_2, b_2]$ 里的一个值，等等；最后的 m_k 位对应区间 $[a_k, b_k]$ 里的一个值。

为初始化一个群体，可以简单地以位的方式随机地设定 *pop_size* 个染色体。如果确实有一些有关最优分布的知识，我们可以使用这些信息来安排初始潜在解的集合。

算法的其他部分是直接的：使用解码后自变量序列的函数值 f 评价每代中的每个染色体，按照适应值的概率分布选择新群体，通过变异和杂交算子改变新群体中的染色体。经过若干代后，如果观察不到更进一步的改进，最好染色体就表示一个可能是全局的最优解。我们通常根据速度和资源判据在一固定数目的循环后停止算法。

对基于适应值的概率分布选择新群体的选择过程，通常使用一个根据适应值调节刻度宽距的轮盘。并按照如下的方法构造这样一个轮盘（假设这里的适应值是正值，若不是，可以使用一些比例机制对它们进行调整，这将在第4章中讨论）：

---

① 译者注：原书为 $\max g(x)=\max\{g(x)+C\}$，其中的“=”这里应该用“→”替换。

- 计算每个染色体 v_i (i=1,..., *pop_size*)的适应值 $eval(v_i)$;
- 计算群体的总适应值:

$$F = \sum_{i=1}^{pop_size} eval(v_i)$$

- 计算每个染色体 v_i (i=1,..., *pop_size*)的选择概率 p_i:

$$p_i = eval(v_i) / F$$

- 计算每个染色体 v_i (i=1,..., *pop_size*)的累积概率 q_i:

$$q_i = \sum_{j=1}^{i} p_j$$

对轮盘转动 *pop_size* 次:每次按照下面的方法为新群体选择一个单个的染色体:

- 产生一个在区间 [0, 1]里的随机浮点数 r;
- 如果 $r < q_1$,选择第一个染色体(v_1);否则选择使 $q_{i-1} < r \le q_i$ 成立的第 i 个染色体 v_i $(2 \le i \le pop_size)$。

很明显,这样的染色体将被选择一次以上。这是符合模式定理的(见下一章):最好染色体得到多个拷贝,中等染色体保持平稳,最差染色体死亡。

现在,我们准备在新群体中的个体应用重组算子 — 杂交。如前所述,遗传系统的一个参数是杂交概率 p_c。此概率给出预计要进行杂交的染色体个数 $p_c \cdot pop_size$。我们进行下面的处理:对新群体中的每个染色体

- 产生一个在区间[0, 1]里的随机浮点数 r;
- 如果 $r < p_c$,选择给定的染色体进行杂交。

随机地对被选择染色体配对:对染色体对中的每一个, 产生一个在区间[1, m−1](*m* 为总长 —— 染色体的位数)里的随机整数 *pos*。 数 *pos* 表示杂交点的位置。两个染色体

$$(b_1 b_2 \ldots b_{pos} b_{pos+1} \ldots b_m) \text{ 和 } (c_1 c_2 \ldots c_{pos} c_{pos+1} \ldots c_m)$$

被它们的子代:

$$(b_1 b_2 \ldots c_{pos} c_{pos+1} \ldots c_m) \text{ 和 } (c_1 c_2 \ldots b_{pos} b_{pos+1} \ldots b_m)$$

所替换。

下一个算子变异,是在一位一位(bit-by-bit)基础上执行的。另一个遗传系统参数,变异率 p_m,给出我们预计的变异位数:$p_m \cdot m \cdot pop_size$。整个群体中所有染色体中的每一位都有均等的机会经历变异,即从 0 到 1 或者相反,所以进行如下的步骤。

对杂交后的当前群体中的每个染色体和染色体中的每个位:

- 产生在区间 [0, 1]里的一随机浮点数 r;
- 如果 $r < p_m$,变异此位。

随着选择、杂交和变异的进行,新群体就为下一次的评价做好了准备。该评价是用来为下一次选择过程建立概率分布的,即建立一个根据当前适应值构造宽距的轮盘。演

化的其余部分只是上述步骤的循环重复，见引言部分的图 0.1。

整个过程可以用一个例子来说明。我们模拟一个遗传算法来进行函数优化。假定群体规模 $pop_size = 20$，遗传算子的概率为 $p_c = 0.25$ 和 $p_m = 0.01$。

问题是求下面函数的最大值：

$$f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

其中 $-3.0 \le x_1 \le 12.1$ 及 $4.1 \le x_2 \le 5.8$。函数 f 如图 2.1 所示。

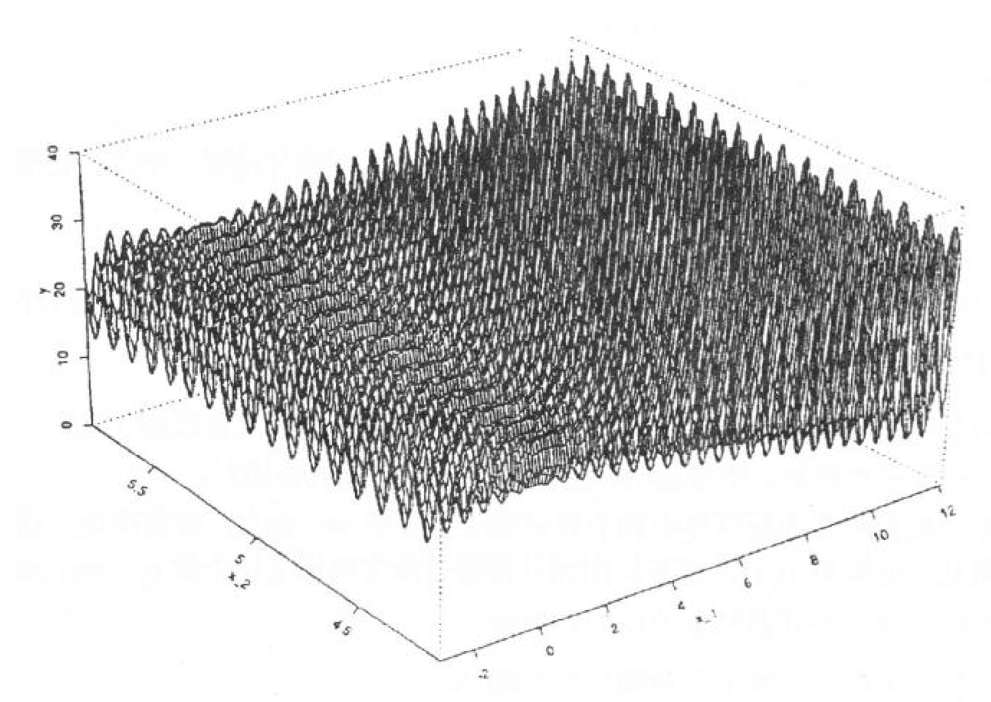


图 2.1 函数 $f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$的图

进一步假定对每个变量要求的精度是小数点以后第四位。变量 x_1 的定义域长度为 15.1：所要求的精度意味着区间[−3.0, 12.1]应该至少被分成 15.1 · 10000 个等距区间，即染色体的第一部分需要 18 位：

$$2^{17} < 151000 < 2^{18}$$

自变量 x_2 域长度为 1.7；精度要求区间 [4.1, 5.8]应该至少被分成 1.7 · 10000 个等距区间，即染色体的第二部分需要 15 位：

$$2^{14} < 17000 \le 2^{15}$$

所以染色体（解向量）的总长度为 $m = 18 + 15 = 33$ 位，即前 18 位的 x_1 和后 15 位的 x_2（19 —33）。考虑染色体：

(010001001011010000111110010100010)

前 18 位 010001001011010000

表示

$$x_1 = -3.0 + decimal(010001001011010000_2) \cdot \frac{121-(-3.0)}{2^{18}-1}$$

$$= -3.0 + 70352 \cdot \frac{15.1}{262143}$$

$$= 1.052426$$

后 15 位 111110010100010

表示

$$x_2 = 4.1 + decimal\,(111110010100010_2) \cdot \frac{5.8 - 4.1}{2^{15} - 1}$$

$$= 4.1 + 31960 \cdot \frac{1.7}{32767}$$

$$= 5.755330$$

所以染色体 (010001001011010000111110010100010)

对应于$\langle x_1, x_2 \rangle = \langle 1.052426, 5.755330 \rangle$。该染色体的适应值为：

$$f(1.052426, 5.755330) = 20.252640$$

用遗传算法的优化函数 f ，产生了一个有 $pop_size = 20$ 个染色体的群体。所有染色体中的 33 位都是随机初始化的。

假定经过初始化过程后，我们得到下面的群体：

v_1= (100110100000001111111010011011111)
v_2= (111000100100110111001010100011010)
v_3= (000010000011001000001010111011101)
v_4= (100011000101101001111000001110010)
v_5= (000111011001010011010111111000101)
v_6= (000101000010010101001010111111011)
v_7= (001000100000110101111011011111011)
v_8= (100001100001110100010110101100111)
v_9= (010000000101100010110000001111100)
v_{10}= (000001111000110000011010000111011)
v_{11}= (011001111110110101100001101111000)
v_{12}= (110100010111101101000101010000000)
v_{13}= (111011111010001000110000001000110)
v_{14}= (010010011000001010100111100101001)
v_{15}= (111011101101110000100011111011110)
v_{16}= (110011110000011111100001101001011)
v_{17}= (011010111111001111010001101111101)
v_{18}= (011101000000001110100111110101101)
v_{19}= (000101010011111111110000110001100)
v_{20}= (101110010110011110011000101111110)

在评价阶段，对每个染色体进行解码并计算解码后的 (x_1, x_2)的适应函数值。得到：

$eval\,(v_1) = f(6.084492, 5.652242) = 26.019600$
$eval\,(v_2) = f(10.348434, 4.380264) = 7.580015$
$eval\,(v_3) = f(-2.516603, 4.390381) = 19.526329$
$eval\,(v_4) = f(5.278638, 5.593460) = 17.406725$
$eval\,(v_5) = f(-1.255173, 4.734458) = 25.341160$
$eval\,(v_6) = f(-1.811725, 4.391937) = 18.100417$
$eval\,(v_7) = f(-0.991471, 5.680258) = 16.020812$

$$eval(\mathbf{v}_8) = f(4.910618, 4.703018) = 17.959701$$
$$eval(\mathbf{v}_9) = f(0.795406, 5.381472) = 16.127799$$
$$eval(\mathbf{v}_{10}) = f(-2.554851, 4.793707) = 21.278435$$
$$eval(\mathbf{v}_{11}) = f(3.130078, 4.996097) = 23.410669$$
$$eval(\mathbf{v}_{12}) = f(9.356179, 4.239457) = 15.011619$$
$$eval(\mathbf{v}_{13}) = f(11.134646, 5.378671) = 27.316702$$
$$eval(\mathbf{v}_{14}) = f(1.335944, 5.151378) = 19.876294$$
$$eval(\mathbf{v}_{15}) = f(11.089025, 5.054515) = 30.060205$$
$$eval(\mathbf{v}_{16}) = f(9.211598, 4.993762) = 23.867227$$
$$eval(\mathbf{v}_{17}) = f(3.367514, 4.571343) = 13.696165$$
$$eval(\mathbf{v}_{18}) = f(3.843020, 5.158226) = 15.414128$$
$$eval(\mathbf{v}_{19}) = f(-1.746635, 5.395584) = 20.095903$$
$$eval(\mathbf{v}_{20}) = f(7.935998, 4.757338) = 13.666916$$

很明显，染色体 $\mathbf{v}_{15}$ 是最好的，染色体 $\mathbf{v}_2$ 是最差的。

现在，系统为选择过程建立一个轮盘。群体的总适应值为

$$F = \sum_{i=1}^{20} eval(\mathbf{v}_i) = 387.776822$$

对每个染色体 $\mathbf{v}_i$ $(i=1,\ldots,pop_size)$，选择概率 p_i 为：

$p_1 = eval(\mathbf{v}_1) / F = 0.067099$　　$p_2 = eval(\mathbf{v}_2) / F = 0.019547$

$p_3 = eval(\mathbf{v}_3) / F = 0.050355$　　$p_4 = eval(\mathbf{v}_4) / F = 0.044889$

$p_5 = eval(\mathbf{v}_5) / F = 0.065350$　　$p_6 = eval(\mathbf{v}_6\}) / F = 0.046677$

$p_7 = eval(\mathbf{v}_7) / F = 0.041315$　　$p_8 = eval(\mathbf{v}_8) / F = 0.046315$

$p_9 = eval(\mathbf{v}_9) / F = 0.041590$　　$p_{10} = eval(\mathbf{v}_{10}) / F = 0.054873$

$p_{11} = eval(\mathbf{v}_{11}) / F = 0.060372$　　$p_{12} = eval(\mathbf{v}_{12}) / F = 0.038712$

$p_{13} = eval(\mathbf{v}_{13}) / F = 0.070444$　　$p_{14} = eval(\mathbf{v}_{14}) / F = 0.051257$

$p_{15} = eval(\mathbf{v}_{15}) / F = 0.077519$　　$p_{16} = eval(\mathbf{v}_{16}) / F = 0.061549$

$p_{17} = eval(\mathbf{v}_{17}) / F = 0.035320$　　$p_{18} = eval(\mathbf{v}_{18}) / F = 0.039750$

$p_{19} = eval(\mathbf{v}_{19}) / F = 0.051823$　　$p_{20} = eval(\mathbf{v}_{20}) / F = 0.035244$

每个染色体 $\mathbf{v}_i$ $(i=1,\ldots,20)$的累积概率 q_i 为：

$q_1 = 0.067099$　　$q_2 = 0.086647$　　$q_3 = 0.137001$　　$q_4 = 0.181890$

$q_5 = 0.247240$　　$q_6 = 0.293917$　　$q_7 = 0.335232$　　$q_8 = 0.381546$

$q_9 = 0.423137$　　$q_{10} = 0.478009$　　$q_{11} = 0.538381$　　$q_{12} = 0.577093$

$q_{13} = 0.647537$　　$q_{14} = 0.698794$　　$q_{15} = 0.776314$　　$q_{16} = 0.837863$

$q_{17} = 0.873182$　　$q_{18} = 0.912932$　　$q_{19} = 0.964756$　　$q_{20} = 1.000000$

现在，准备转动轮盘 20 次；每次为新群体选择一单个的染色体。假定在区间[0, 1]里的 20 个数的一个随机序列是：

0.513870	0.175741	0.308652	0.534534	0.947628
0.171736	0.702231	0.226431	0.494773	0.424720
0.703899	0.389647	0.277226	0.368071	0.983437

0.005398	0.765682	0.646473	0.767139	0.780237

第一个数 $r = 0.513870$ 大于 q_{10} 小于 q_{11}，意味着染色体 v_{11} 被选择；第二个数 $r = 0.175741$ 大于 q_3 且小于 q_4，意味着染色体 v_4 被选择，等等。

最后，新群体由下面的染色体组成：

v'_1 = (011001111110110101100001101111000)　(v_{11})
v'_2 = (100011000101101001111000001110010)　(v_4)
v'_3 = (001000100000110101111011011111011)　(v_7)
v'_4 = (011001111110110101100001101111000)　(v_{11})
v'_5 = (000101010011111111110000110001100)　(v_{19})
v'_6 = (100011000101101001111000001110010)　(v_4)
v'_7 = (111011101101110000100011111011110)　(v_{15})
v'_8 = (000111011001010011010111111000101)　(v_5)
v'_9 = (011001111110110101100001101111000)　(v_{11})
v'_{10} = (000010000011001000001010111011101)　(v_3)
v'_{11} = (111011101101110000100011111011110)　(v_{15})
v'_{12} = (010000000101100010110000001111100)　(v_9)
v'_{13} = (000101000010010101001010111111011)　(v_6)
v'_{14} = (100001100001110100010110101100111)　(v_8)
v'_{15} = (101110010110011110011000101111110)　(v_{20})
v'_{16} = (100110100000001111111010011011111)　(v_1)
v'_{17} = (000001111000110000011010000111011)　(v_{10})
v'_{18} = (111011111010001000110000001000110)　(v_{13})
v'_{19} = (111011101101110000100011111011110)　(v_{15})
v'_{20} = (110011110000011111100001101001011)　(v_{16})

下面准备对新群体（向量 v_i）中的个体应用杂交算子。杂交概率 $p_c = 0.25$，所以预计染色体中平均有 25%（即 20 个中的 5 个）将经历杂交。杂交按照下面的方法进行：对新群体中的每个染色体，产生一在区间[0, 1]里的随机数 r，如果 $r < 0.25$，则选择一给定的染色体进行杂交。假定随机数序列是：

0.822951	0.151932	0.625477	0.314685	0.346901
0.917204	0.519760	0.401154	0.606758	0.785402
0.031523	0.869921	0.166525	0.674520	0.758400
0.581893	0.389248	0.200232	0.355635	0.826927

这就意味着染色体 v_2，v_{11}，v_{13} 和 v_{18} 被选择杂交。这里很幸运：给选择的染色体数是偶数，可以很容易地配对。如果选择的染色体数为奇数，可以加入一额外的染色体或者移走一被选择染色体，这种选择同样是随机的。对被选择染色体随机进行配对：设为前两个（即 v'_2 和 v'_{11}）和后两个（即 v'_{13} 和 v'_{18}）被配对。对这两对中的每一对，产生

区间[1, 32]（33 为总长 —— 染色体中的位数）里的一个随机整数 *pos*。数字 *pos* 表示杂交点的位置。第 1 对染色体是：

$$v_2' = (100011000|101101001111000001110010)$$
$$v_{11}' = (111011101|101110000100011111011110)$$

产生的数为 $pos = 9$。这对染色体在第 9 位后被分割并被它们的子代所替换：

$$v_2'' = (100011000|101110000100011111011110)$$
$$v_{11}'' = (111011101|101101001111000001110010)$$

第二对染色体是

$$v_{13}' = (00010100001001010100|1010111111011)$$
$$v_{18}' = (11101111101000100011|0000001000110)$$

产生的数字 $pos = 20$。这对染色体被它们的子代所替换：

$$v_{13}'' = (00010100001001010100|0000001000110)$$
$$v_{18}'' = (11101111101000100011|1010111111011)$$

群体的当前版本为：

$$v_1' = (011001111110110101100001101111000)$$
$$v_2'' = (100011000101110000100011111011110)$$
$$v_3' = (001000100000110101111011011111011)$$
$$v_4' = (011001111110110101100001101111000)$$
$$v_5' = (000101010011111111110000110001100)$$
$$v_6' = (100011000101101001111000001110010)$$
$$v_7' = (111011101101110000100011111011110)$$
$$v_8' = (000111011001010011010111111000101)$$
$$v_9' = (011001111110110101100001101111000)$$
$$v_{10}' = (000010000011001000001010111011101)$$
$$v_{11}'' = (111011101101101001111000001110010)$$
$$v_{12}' = (010000000101100010110000001111100)$$
$$v_{13}'' = (000101000010010101000000001000110)$$
$$v_{14}' = (100001100001110100010110101100111)$$
$$v_{15}' = (101110010110011110011000101111110)$$
$$v_{16}' = (100110100000001111111010011011111)$$
$$v_{17}' = (000001111000110000011010000111011)$$
$$v_{18}'' = (111011111010001000111010111111011)$$
$$v_{19}' = (111011101101110000100011111011110)$$
$$v_{20}' = (110011110000011111100001101001011)$$

下一个算子变异，是在一位一位基础上执行的。变异概率 $p_m = 0.01$，所以我们预计平均将有 1%的位经历变异。整个群体共有 $m \times pop_size = 33 \times 20 = 660$ 位；可以预计平

均每代有 6.6 次变异。因为每一位都有均等的机会被变异，所以对群体中的每一位可以产生区间[0, 1]里的--个随机数 r，如果 $r < 0.01$，则变异此位。

这就意味着我们不得不产生 660 个随机数。在运行例子中，共产生 5 个小于 0.01 的数；变异位号和产生的随机数如表 2.1 所示。

表 2.1 运行例子中产生的小于 0.01 的随机数及其相应的位号

位号	随机数
112	0.000213
349	0.009945
418	0.008809
429	0.005425
602	0.002836

下表 2.2 将染色体中的位号翻译成染色体中的位数。

表 2.2 位号及其相应的染色体序号和染色体内位序号

位号	染色体序号	染色体内位序号
112	4	13
349	11	19
418	13	22
429	13	33
602	19	8

这就意味着有四个染色体通过变异而改变：其中一个染色体（第 13 个）有两位发生了变化。

变异位以黑体表示。对修正的染色体去掉撇号，最新的向量群体 v_i 如下：

$$v_1 = (011001111110110101100001101111000)$$
$$v_2 = (100011000101110000100011111011110)$$
$$v_3 = (001000100000110101111011011111011)$$
$$v_4 = (011001111110010101100001101111000)$$
$$v_5 = (000101010011111111110000110001100)$$
$$v_6 = (100011000101101001111000001110010)$$
$$v_7 = (111011101101110000100011111011110)$$
$$v_8 = (000111011001010011010111111000101)$$
$$v_9 = (011001111110110101100001101111000)$$
$$v_{10} = (000010000011001000001010111011101)$$
$$v_{11} = (111011101101101001011000001110010)$$
$$v_{12} = (010000000101100010110000001111100)$$
$$v_{13} = (000101000010010101000100001000111)$$
$$v_{14} = (100001100001110100010110101100111)$$
$$v_{15} = (101110010110011110011000101111110)$$
$$v_{16} = (100110100000001111111010011011111)$$
$$v_{17} = (000001111000110000011010000111011)$$
$$v_{18} = (111011111010001000111010111111011)$$
$$v_{19} = (111011100101110000100011111011110)$$

v_{20} = (110011110000011111100001101001011)

我们这里只是完成了遗传过程 **while** 循环中的一次迭代（即一代），见引言中的图 0.1。检查新群体的评价过程是很有趣的。在评价阶段，对每个染色体进行解码，并计算解码后的(x_1,x_2)的适应函数值。得到：

eval (v_1) = f(3.130078, 4.996097) = 23.410669
eval (v_2) = f(5.279042, 5.054515) = 18.201083
eval (v_3) = f(−0.991471, 5.680258) = 16.020812
eval (v_4) = f(3.128235, 4.996097) = 23.412613
eval (v_5) = f(−1.746635, 5.395584) = 20.095903
eval (v_6) = f(5.278638, 5.593460) = 17.406725
eval (v_7) = f(11.089025, 5.054515) = 30.060205
eval (v_8) = f(−1.255173, 4.734458) = 25.341160
eval (v_9) = f(3.130078, 4.996097) = 23.410669
eval (v_{10}) = f(−2.516603, 4.390381) = 19.526329
eval (v_{11}) = f(11.088621, 4.743434) = 33.351874
eval (v_{12}) = f(0.795406, 5.381472) = 16.127799
eval (v_{13}) = f(−1.811725, 4.209937) = 22.692462
eval (v_{14}) = f(4.910618, 4.703018) = 17.959701
eval (v_{15}) = f(7.935998, 4.757338) = 13.666916
eval (v_{16}) = f(6.084492, 5.652242) = 26.019600
eval (v_{17}) = f(−2.554851, 4.793707) = 21.278435
eval (v_{18}) = f(11.134646, 5.666976) = 27.591064
eval (v_{19}) = f(11.059532, 5.054515) = 27.608441
eval (v_{20}) = f(9.211598, 4.993762) = 23.867227

注意新群体的总适应值 F 为 447.049688，高于先前群体的总适应值 387.776822。同时，当前最好染色体 v_{11} 的评价值(33.351874)要好于先前群体的最好染色体 v_{15} 的评价值(30.060205)。

现在准备再一次运行选择过程：应用遗传算子及评价下一代等。经过 1000 代后，群体为：

v_1 = (111011110110011011100101010111011)
v_2 = (111001100110000100010101010111000)
v_3 = (111011110111011011100101010111011)
v_4 = (111001100010000110000101010111001)
v_5 = (111011110111011011100101010111011)
v_6 = (111001100110000100000100010100001)
v_7 = (110101100010010010001100010110000)
v_8 = (111101100010001010001101010010001)
v_9 = (111001100010010010001100010110001)

v_{10}= (111011110111011011100101010111011)

v_{11}= (110101100000010010001100010110000)

v_{12}= (110101100010010010001100010110001)

v_{13}= (111011110111011011100101010111011)

v_{14}= (111001100110000100000101010111011)

v_{15}= (111001101010111001010100110110001)

v_{16}= (111001100110000101000100010100001)

v_{17}= (111001100110000100000101010111011)

v_{18}= (111001100110000100000101010111001)

v_{19}= (111101100010001010001110000010001)

v_{20}= (111001100110000100000101010111001)

相应的适应值为：

$eval(v_1) = f(11.120940, 5.092514) = 30.298543$

$eval(v_2) = f(10.588756, 4.667358) = 26.869724$

$eval(v_3) = f(11.124627, 5.092514) = 30.316575$

$eval(v_4) = f(10.574125, 4.242410) = 31.933120$

$eval(v_5) = f(11.124627, 5.092514) = 30.316575$

$eval(v_6) = f(10.588756, 4.214603) = 34.356125$

$eval(v_7) = f(9.631066, 4.427881) = 35.458636$

$eval(v_8) = f(11.518106, 4.452835) = 23.309078$

$eval(v_9) = f(10.574816, 4.427933) = 34.393820$

$eval(v_{10}) = f(11.124627, 5.092514) = 30.316575$

$eval(v_{11}) = f(9.623693, 4.427881) = 35.477938$

$eval(v_{12}) = f(9.631066, 4.427933) = 35.456066$

$eval(v_{13}) = f(11.124627, 5.092514) = 30.316575$

$eval(v_{14}) = f(10.588756, 4.242514) = 32.932098$

$eval(v_{15}) = f(10.606555, 4.653714) = 30.746768$

$eval(v_{16}) = f(10.588814, 4.214603) = 34.359545$

$eval(v_{17}) = f(10.588756, 4.242514) = 32.932098$

$eval(v_{18}) = f(10.588756, 4.242410) = 32.956664$

$eval(v_{19}) = f(11.518106, 4.472757) = 19.669670$

$eval(v_{20}) = f(10.588756, 4.242410) = 32.956664$

但是，如果仔细检查整个运行过程，可以发现早期代中的某些染色体的适应值要好于经过 1000 代后得到的最好染色体值 35.477938。例如，第 396 代中的最好染色体的值为 38.827553。这是由于取样的随机误差造成的，我们将在第 4 章讨论这个问题。

跟踪进化过程中的最好个体是很容易的。在遗传算法的实现中，通常在一独立出来的位置保存“曾经最好”个体；用这种方法，算法将报告整个过程的最好值，而不只是最终群体的最好值。

第 3 章　遗传算法的理论基础

遗传算法的理论基础是遗传算法解的二进制表达式及模式的含义。模式[188]是能对染色体之间相似性进行解释的模板。一个模式是通过引入通配符（*）到基因字母表中来建立的。一个模式代表所有的除“*”外与所有位置都匹配的所有串（是超平面或搜索空间的子集）。例如，考虑长度为 10 的串和模式。模式(* 1 1 1 1 0 0 1 0 0)与两个串匹配：

{(0111100100), (1111100100)}

模式(*1*1 1 0 0 1 0 0)与四个串匹配：

{(0101100100), (0111100100), (1101100100), (1111100100)}

当然，模式(1 0 0 1 1 1 0 0 0 1)只代表一个串：(1001110001)，模式(**********)代表所有长度为 10 的串。很明显，每种模式精确地代表 2^r 个串，这里 r 为通配符“*”在模式模板中的个数。另一方面，每个长度为 m 的串匹配 2^m 个模式。如串(1001110001)匹配下面 2^{10} 个模式：

(1 0 0 1 1 1 0 0 0 1)
(* 0 0 1 1 1 0 0 0 1)
(1 * 0 1 1 1 0 0 0 1)
(1 0 * 1 1 1 0 0 0 1)
…………
(1 0 0 1 1 1 0 0 0 *)
(* * 0 1 1 1 0 0 0 1)
(* 0 * 1 1 1 0 0 0 1)
…………
(1 0 0 1 1 1 0 0 * *)
(* * * 1 1 1 0 0 0 1)
…………
(* * * * * * * * * *)

考虑长度为 m 的串，共有 3^m 种可能的模式。若群体规模为 n，则可能有在 2^m 和 $n \cdot 2^m$ 之间种不同的模式。

不同的模式有不同的特性。我们已经注意到：在一个模式中通配符 * 的数目决定了匹配该模式的串数。这里有两个重要的模式性质：阶和定义长度；模式定理是在这些性质的基础上表达的。

模式的阶 S(由 $o(S)$表示)，为串中 0 或 1 的数目，即固定位而非通配位的数目。换句话说，它为模板长度减去通配符（*）的数目。一个模式的阶定义了模式的特殊性。如下面三个长度为 10 的模式

$$S_1 = (* * * 0 0 1 * 1 1 0)$$

$$S_2 = (* * * * 0 0 * * 0 *)$$

$$S_3 = (1 1 1 0 1 * * 0 0 1)$$

的阶为：

$$o(S_1) = 6,\ o(S_2) = 3 \text{ 及 } o(S_3) = 8$$

其中模式 S_3 是最特殊的一个。

模式的阶对计算变异操作中模式的存活概率是很有用的。我们将在本章后面讨论。

模式的定义长度 S（由$\delta(S)$表示），为第一个和最后一个固定串位之间的距离。它定义了包含在模式中的信息的致密度。如，

$$\delta(S_1) = 10 - 4 = 6,\quad \delta(S_2) = 9 - 5 = 4 \text{ 和} \delta(S_3) = 10 - 1 = 9$$

注意：只有单个固定点的模式的定义长度为零。

模式的定义长度在计算模式杂交的存活概率时是很有用的，这将在本章后面讨论。

正如前边所说，遗传算法模拟进化过程包括四个不断重复的步骤：

$t \leftarrow t + 1$

select $P(t)$ from $P(t-1)$

recombine $P(t)$

evaluate $P(t)$

第一步($t \leftarrow t + 1$) 简单地将演化时钟向前移动一个单位；最后一步[evaluate $P(t)$]只是对当前群体进行评价。发生在进化周期其余两个阶段的主要现象是选择和重组。我们将讨论这两步作用在群体中预定数目的模式上的效果。首先从选择步骤开始，通过运行一个例子来说明所有的定理。

假定群体规模 *pop_size* = 20，串长，即模式模板的长度 m=33（和前一章所讨论的运行例子相同）。并进一步假定在时刻 t，群体由下面的串组成：

v_1 = (100110100000001111111010011011111)

v_2 = (111000100100110111001010100011010)

v_3 = (000010000011001000001010111011101)

v_4 = (100011000101101001111000001110010)

v_5 = (000111011001010011010111111000101)

v_6 = (000101000010010101001010111111011)

v_7 = (001000100000110101111011011111011)

v_8 = (100001100001110100010110101100111)

v_9 = (010000000101100010110000001111100)

v_{10} = (000001111000110000011010000111011)

v_{11} = (011001111110110101100001101111000)

v_{12} = (110100010111101101000101010000000)

v_{13} = (111011111010001000110000001000110)

v_{14} = (010010011000001010100111100101001)

v_{15} = (111011101101110000100011111011110)

$v_{16}=(110011110000011111100001101001011)$
$v_{17}=(011010111111001111010001101111101)$
$v_{18}=(011101000000001110100111110101101)$
$v_{19}=(000101010011111111110000110001100)$
$v_{20}=(101110010110011110011000101111110)$

$\xi(S,t)$表示时刻 t 时群体中匹配模式 S 的串数，如对一个给定模式：

$$S_0=(****111**************************)$$

$\xi(S_0,t)=3$，因为有 3 个串，即 v_{13}、v_{15} 和 v_{16} 匹配模式 S_0。注意模式 S_0 的阶 $o(S_0)=3$，其定义长度ξ $(S_0)=7-5=2$。

模式的另一个性质是在时刻 t 的适应值 $eval$ (S,t)。它定义为群体中和模式 S 匹配的所有串的平均适应值。假定时刻 t，群体中匹配模式 S 的串有 p 个$\{v_{i_1},\ldots,v_{i_p}\}$，则

$$eval(S,t)=\sum_{j=1}^{p} eval(v_{i_j})/p$$

在选择过程中，中间群体是这样产生的：对 $pop_size=20$ 个单个串进行选择。每个串根据其适应值被拷贝零次、一次或多次。正如在先前例子中所看到的，在一个单个串的选择中，串 v_i 被选择的概率 $p_i=eval(v_i)/F(t)$，其中 $F(t)$为整个群体在时刻 t 的总适应值

$$F(t)=\sum_{i=1}^{20} eval(v_i)$$

经过选择步骤，我们预计有 $\xi(S,t+1)$ 个串被模式 S 匹配。因为（1）对一个被模式 S 匹配的一般的串，在一个单个串选择中，其选择概率等于 $eval(S,t)/F(t)$；（2）被模式 S 匹配的串数为 $\xi(S,t)$；（3）单个串的选择数目为 pop_size，很明显

$$\xi(S,t+1)=\xi(S,t)\cdot pop_size\cdot eval(S,t)/F(t)$$

重写上述公式：考虑群体的平均适应值 $\overline{F(t)}=F(t)/pop_size$，上式可以写成：

$$\xi(S,t+1)=\xi(S,t)\cdot eval(S,t)/\overline{F(t)} \tag{3.1}$$

换句话说，群体的串生长的数目是模式适应值与群体平均适应值之比。这就意味着一个适应值在平均之上的模式在下一代中的串数会增加，一个适应值低于平均的模式在下一代中的串数会减少，而一个中庸的模式将保持不变的水平。

上述规则的长期效果是很明显的。如果假定模式 S 保持高出平均 $\varepsilon\%$，即

$$eval(S,t)=\overline{F(t)}+\varepsilon\cdot\overline{F(t)}$$

那么 $\xi(S,t)=\xi(S,0)(1+\varepsilon)^t$

且 $\varepsilon=(eval(S,t)-\overline{F(t)})/\overline{F(t)}$

其中，$\varepsilon>0$ 相应于平均之上的模式；$\varepsilon<0$ 相应于平均之下的模式。

这是一个几何级数方程：一个平均之上的模式不仅在下一代中串数增长，而且在下一代中出现一个串数几何增长。

我们把这个方程（3.1）叫做复制模式增长方程。

现在再回到例子中的模式 S_0。因为在时刻 t 有 3 个串，即 v_{13}、v_{15} 和 v_{16} 被模式 S_0

匹配，模式的适应值 $eval(S_0)$为：

$$eval(S_0, t) = (27.316702 + 30.060205 + 23.867227) / 3 = 27.081378$$

同时，整个群体的平均适应值为：

$$\overline{F(t)} = \sum_{i=1}^{20} eval(v_i) / pop_size = 387.776822 / 20 = 19.388841$$

模式 S_0 的适应值与群体平均适应值的比例为：

$$eval(S_0, t) / \overline{F(t)} = 1.396751$$

这就意味着如果模式 S_0 保持在平均之上，那么它将在下一代中保持串数的几何增长。特别是，如果模式 S_0 保持在平均之上的因子为 1.396751，那么在时刻 $t+1$，我们预计可得到 $3\times1.396751 = 4.19$ 个串（可能有 4 或 5 个）被模式 S_0 匹配，在时刻 $t+2$，有 $3\times1.396751^2 = 5.85$ 这样的串，即可能有 6 个串，等等。

直觉告诉我们：S_0 的模式定义了搜索空间中极有希望的一部分，并且它以几何增长的方式被复制。

现在，我们检查对例子中模式 S_0 的预测。对时刻 t 的群体，模式 S_0 被 3 个串匹配，v_{13}、v_{15} 和 v_{16}。我们用和前一章相同的群体模拟了选择过程。新的群体由下面的染色体组成：

v_1' = (011001111110110101100001101111000)　(v_{11})
v_2' = (100011000101101001111000001110010)　(v_4)
v_3' = (001000100000110101111011011111011)　(v_7)
v_4' = (011001111110110101100001101111000)　(v_{11})
v_5' = (000101010011111111110000110001100)　(v_{19})
v_6' = (100011000101101001111000001110010)　(v_4)
v_7' = (111011101101110000100011111011110)　(v_{15})
v_8' = (000111011001010011010111111000101)　(v_5)
v_9' = (011001111110110101100001101111000)　(v_{11})
v_{10}' = (000010000011001000001010111011101)　(v_3)
v_{11}' = (111011101101110000100011111011110)　(v_{15})
v_{12}' = (010000000101100010110000001111100)　(v_9)
v_{13}' = (000101000010010101001010111111011)　(v_6)
v_{14}' = (100001100001110100010110101100111)　(v_8)
v_{15}' = (101110010110011110011000101111110)　(v_{20})
v_{16}' = (111001100110000101000100010100001)　(v_1)
v_{17}' = (111001100110000100000101010111011)　(v_{10})
v_{18}' = (111011111010001000110000001000110)　(v_{13})
v_{19}' = (111011101101110000100011111011110)　(v_{15})
v_{20}' = (110011110000011111100001101001011)　(v_{16})

的确，模式 S_0 在 $t+1$ 时刻匹配 5 个串：v_7'，v_{11}'，v_{18}'，v_{19}'和 v_{20}'。

然而，从选择空间考虑，选择本身不能产生新的点（潜在解），而只是从中间群体

中复制了一些串。所以在进化周期的第二步，重组向群体引入了新个体。这是通过杂交和变异两个遗传算子完成的。我们将依次讨论这两个算子作用于群体中模式的效果。

首先从杂交开始，考虑下面的例子：正如在本章前面所讨论的，群体中的一个单独的串，设为：v_{18}'

(111011111010001000110000001000110)

被 2^{30} 种模式匹配；该串同样被下面两个模式匹配：

$S_0 = (* * * * 1 1 1 *)$

$S_1 = (1 1 1 * 1 0)$

进一步假定上述串被选择用来杂交（如第 2 章所述）。根据第 2 章的实验（向量 v_{18}' 和 v_{13}'被选择进行杂交），假定杂交位置为 pos = 20。很明显，这样杂交后，模式 S_0 生存下来，即有一个后代仍然匹配 S_0。理由是杂交在一个后代串上保存了第 5，6，7 位上的序列 “111”：

$v_{18}' = (11101111101000100011 | 0000001000110)$

$v_{13}' = (000101000001001010100 | 1010111111011)$

将产生

$v_{18}'' = (11101111101000100011 | 1010111111011)$

$v_{13}'' = (000101000001001010100 | 0000001000110)$

另一方面，模式 S_1 将遭到破坏：子代不与它匹配。理由是在模板开始的固定位“111”和结尾的固定位“10”分布在不同的后代上。

很明显，模式的定义长度对其生存和毁坏的概率起着很重要的作用。注意模式 S_0 的定义长度为$\delta(S_0) = 2$，模式 S_1 的定义长度为$\delta(S_1) = 32$。

通常，杂交位是在 m–1 种可能的位置上被唯一地选择的。这就意味着模式 S 的消亡概率为：

$$p_d(S) = \frac{\delta(S)}{m-1}$$

因此，该模式的存活概率为：

$$p_s(S) = 1 - \frac{\delta(S)}{m-1}$$

实际上，我们的例子中模式 S_0 和 S_1 的存活概率和消亡概率为：

$p_d(S_0) = 2/32$，　　$p_s(S_0) = 30/32$

$p_d(S_1) = 32/32 = 1$，　$p_s(S_1) = 0$

所以结果是可预测的。

有一点很重要：只有一些染色体经历杂交，并且杂交的选择性概率为 p_c。这就意味着一个模式存活的概率实际上为：

$$p_s(S) = 1 - p_c \cdot \frac{\delta(S)}{m-1}$$

再一次参考前面例子中的模式 S_0 ($p_c = 0.25$):

$$p_s(S_0) = 1 - 0.25 \cdot \frac{2}{32} = 63/64 = 0.984375$$

注意，尽管杂交位置是在一个模式的固定位置之间选择的，该模式仍然有机会存活。例如，如果串 v_{18}' 和串 v_{13}'都是以 “111” 开始，“10” 结尾，模式 S_1 将杂交存活，当然，这种可能性十分小。因此，我们需要修正模式存活概率公式：

$$p_s(S) \geq 1 - p_c \cdot \frac{\delta(S)}{m-1}$$

所以，选择和杂交的结合给出我们一个新形式的复制模式生长公式：

$$\xi(S, t+1) \geq \xi(S, t) \cdot eval(S, t) \cdot \left[1 - p_c \frac{\delta(S)}{m-1}\right] / \overline{F(t)} \tag{3.2}$$

公式(3.2)告诉我们在下一代中匹配模式 S 的预测串数是实际匹配模式的串数、模式的相对适应值及模式的定义长度的函数。很明显，在平均之上，短的定义长度的模式将按照几何增长的速率被复制。对模式 S_0：

$$eval(S, t) \cdot \left[1 - p_c \frac{\delta(S)}{m-1}\right] / \overline{F(t)} = 1.396751 \cdot 0.984375 = 1.374927$$

这就意味着短的、平均之上的模式 S_0 将在下一代中出现几何增长的串数：在时刻 $(t+1)$，我们预测将有 $3\times1.374927 = 4.12$ 个串被模式 S_0 匹配（略小于 4.19——我们只考虑选择时得到的值）；在时刻 $(t+2)$，将有 $3\times1.374927^2 = 5.67$ 个这样的串（仍然小于 5.85）。

下一个被考虑的算子是变异。变异算子以概率 p_m 随机地改变一个染色体中的某一位。其变化为从 0 到 1 或相反。很明显，如果模式经过变异还存活，该模式上的所有固定位应保持不变。再一次考虑群体中的一个串，设为 v_{19}'：

(111011101101110000100011111011110)

和模式 S_0：

S_0 = (* * * * 1 1 1 *)

进一步假定串 v_{19}' 经历变异，即至少有一位发生如前一章所示的变化。我们可以回忆起有四个串经历过这样的变异：一个串 v_{13}'经历了两个位的变异，其余三个串经历了一位的变异。因为 v_{19}'在第 9 位变异，其子代：

v_{19}'' = (111011100101110000100011111011110)

仍然被模式 S_0 匹配。如果被选择变异的位置是从 1 到 4，或从 8 到 33，那么所生成的子代将仍然被模式 S_0 匹配。这里只有 3 位（第 5，6 和 7 位——模式 S_0 的固定位）是很重要的：变异其中之一将破坏模式 S_0。很明显，这些重要位的数目等于模式的阶，即固定位的数目。

因为单个位变换的概率为 p_m，单个位存活的概率为 $1-p_m$。单个变异是和其他变异相互独立的，所以模式 S 经过单点变异后的存活概率为：

$$p_s(S) = (1-p_m)^{o(S)}$$

因为 $p_m << 1$，所以此概率可被近似为：

$$p_s(S) \approx 1 - o(S) \cdot p_m$$

再一次参考例子中的模式 S_0 (p_m= 0.01):

$$p_s(S_0) \approx 1-3 \cdot 0.01 = 0.97$$

选择、杂交和变异的组合给出我们复制模式生长公式的新形式:

$$\xi(S, t+1) \geq \xi(S, t) \cdot eval(S, t) \cdot \left[1 - p_c \frac{\delta(S)}{m-1} - o(S) \cdot p_m\right] / \overline{F(t)} \tag{3.3}$$

正如简单公式(3.1)和(3.2),公式(3.3)告诉我们在下一代中,预测匹配模式 S 的串数为匹配模式的实际串数、模式的相对适应值和其定义长度及阶的函数。同样很明显,平均之上的、有短的定义长度和低阶的模式将按照几何增长的速率被复制。

对模式 S_0:

$$eval(S_0, t) \cdot \left[1 - p_c \frac{\delta(S)}{m-1} - o(S) \cdot p_m\right] / \overline{F(t)} = 1.396751 \cdot 0.954375 = 1.333024$$

这就意味着短的、低阶、平均之上的模式 S_0 将在下一代中产生几何增长的串数:在时刻 (t+1),我们预测将有 3×1.333024 = 4.00 个串被模式 S_0 匹配(稍小于 4.19 ——只考虑选择的结果,或 4.12 ——只考虑选择和杂交的结果),在时刻 (t+2),有 $3 \times 1.333024^2 = 5.33$ 个这样的串(仍小于 5.85 或 5.67)。

注意,公式(3.3)是基于适应值函数返回 f 正值的假定:当用遗传算法优化可能返回负值的优化函数时,需要附加一些适应值函数和优化函数之间的映射。我们将在下一章讨论。

概括起来,生长公式(3.1)展示:选择增加了平均之上的模式的复制率,并且这种变化是指数型的。复制本身并不能增加新的模式(初始 t = 0 时的复制除外)。这就是为什么要引入杂交算子的原因——能以结构化的方式随机地交换信息。另外,变异算子向群体引入了较大的变化性。如果一个模式是短的且低阶的,这些算子作用在该模式上的组合分裂效果是不大的。生长公式(3.3)的最终结果可以用下面的定理和假设说明。

定理 1. 模式定理(Schema Theorem.):短的、低阶、平均之上的模式在遗传算法的后续代中将按几何级数增长。

这一理论的直接结果是:遗传算法通过短的、低阶模式的杂交信息交换过程探求了搜索空间。

假设 1. 基因块假设(Building Block Hypothesis):遗传算法是通过并列短的、低阶、高效模式(称之为基因块)来寻求接近最优的执行效果。

正如[154]中所陈述的:

"正如小孩通过摆放简单的积木建造一个大的堡垒一样,遗传算法是通过并置短的、低阶、高效的模式来寻找接近最优的执行效果"

在本章中,我们已经看到了一个极好的基因块的例子:

$$S_0 = (****111****************************)$$

S_0 是一个短的、低阶、至少在初期群体中是平均之上的模式。该模式朝着寻求最优的方向演化。

虽然已有一些研究来证明此假设[38],多数重要的应用还是依赖于实验结果。在过去

15 年中，很多遗传算法在众多问题领域里的应用支持基因块假设。毫无疑问，基因块假设说明遗传算法的编码问题对其执行效果来说是至关重要的，并且这种编码应该符合短基因块的思想。

在本章的前边，我们就已论述了一个群体规模为 *pop_size*，长度为 m 的个体至少有 2^m 个模式，至多有 2^{pop_size} 个模式。它们中的一些以一种有益的方式被处理：以合意的指数增长率被复制，并不被杂交和变异所分裂（分裂有可能发生在长定义长度和高阶的模式上）。

Holland[188]阐明：它们中至少有 pop_size^3 个被有意义地处理——Holland 称之为隐含并行性。因为它不需额外的内存或额外的处理就可获得。很有趣的是，一个有 *pop_size* 个串的群体有远远超过 *pop_size* 个模式①。

在本章，我们提供了遗传算法工作原理的一些标准解释。注意，基因块假设仅仅是一种思想上的解释，在某些具体问题上很容易发生背离。如，假定有两个总长为 11 的、短的、低阶模式：disirable

$$S_1 = (1\,1\,1 * * * * * * * *) \text{ 和 } S_2 = (* * * * * * * * * 1\,1)$$

是平均之上的，但它们的组合

$$S_3 = (1\,1\,1 * * * * * * 1\,1)$$

要比　$S_4 = (0\,0\,0 * * * * * * 0\,0)$

更不适。进一步假定优化串为 S_3 所匹配的 S_0 = (11111111111) 。遗传算法可能难以收敛到 S_0，因为它可能趋向于收敛到像(00011111100)这样的点。这种现象叫欺骗(deception)[38, 154]：一些短的、低阶的基因块可能误导遗传算法，使其收敛到次最优点。

欺骗现象是和上位（epistasis）的概念紧密相连的，用遗传算法的术语，上位衡量着染色体基因之间的强相互作用②。换句话说，上位度量一个基因的适应值依赖与另一个基因适应值的程度。对一个给定问题，高的上位度意味着基因块不能形成，因此该问题是欺骗问题。

建议有三种方法来解决欺骗问题[155]。第一种方法是将目标函数先前的知识用合适的方法进行编码以获得紧凑的基因块。例如，先是有关目标函数的先验知识，然后是欺骗问题，可能产生不同的编码方式，如要求优化的函数是五位相邻的而不是六位分开的。

第二种方法是使用第三种遗传算子转置（inversion）。简单的转置是一个像变异一样的算子：它在一个串内选择两点并在两点之间倒置位的次序，但是要记住位的“意义”。这就意味着我们必须识别串中的位：将位与它们的原始位置记录连在一起就可做到。如：一个串

$$s = ((1,0)(2,0)(3,0)|(4,1)(5,1)(6,0)(7,1)|(8,0)(9,0)(10,0)(11,1))$$

有两个标记点，经过转置后变为

$$s' = ((1,0)(2,0)(3,0)|(7,1)(6,0)(5,1)(4,1)|(8,0)(9,0)(10,0)(11,1))$$

① 最近，Bertoni 和 Dorigo[46] 的研究表明：只有在特殊的情况下，即当 *pop_size* 与 2^l 成比例的情况下，pop_size^3 的估计才是正确的，他们对此进行了更一般性的分析。

② 遗传学家用“上位”这个术语指掩盖或转换效应：如果一个基因的存在压制了另一个位置上基因的效能，则称该基因是上位的。

带有转置算子的遗传算法通过搜索最好的位安排来构造基因块。如先前想要的模式:

$$S_3 = (1\ 1\ 1\ *\ *\ *\ *\ *\ *\ 1\ 1)$$

可重写成

$$S_3 = ((1,1)(2,1)(3,1)(4,*)(5,*)(6,*)(7,*)(8,*)\ (9,*)(10,1)(11,1))$$

经过转置后可能重组为

$$S_3 = ((1,1)(2,1)(3,1)(11,1)(10,1)\ (9,*)(8,*)\ (7,*)(6,*)(5,*)(4,*))$$

这样就构造了一个重要的基因块。然而,正如[155]里所描述的:

> "早期的研究[160]认为一元算子转置不能有效地搜索到紧凑的基因块,因为它缺乏二元算子所固有的交叉重叠能力。换句话说,转置针对的是次序,正如变异是针对等位基因一样:它们都是为了不让缺乏多样性的搜索停止,但是当好的结构要求自身个体部分的上位迭代时,它们又都不足以搜索到好的结构、等位基因或排列。"

最近建议的第三种克服欺骗问题的方法[155,159]是散乱遗传算法(mGA)。因为 mGA 有一些其他有趣的性质,我们将在下一章第 4.6 节对它进行简要的讨论。

第4章 遗传算法的典型专题

遗传算法理论解释了为什么对一个给定问题表达，能收敛到欲求的最优点。不幸的是，实际的应用并不总是遵循这一理论，主要的原因是：

- 对问题的编码经常使遗传算法在不同于问题本身的空间运行。
- 理论假定迭代次数是无限的，而实际上有限制。
- 理论假定群体规模是无限的，而实际上有限制。

这说明遗传算法在某种条件下不能找到最优解：这种失败是由于过早收敛到局部最优造成的。过早收敛是遗传算法和其他优化算法共同的问题。如果收敛发生得太快，包含在群体中的有价值的信息常常会失去。而遗传算法的执行趋向于在找到最优解前过早收敛，正如参考文献[46]中所述：

> "……当遗传算法在多数情况下执行得比其他搜索技术好时，遗传算法仍然难以达到理论上所希望的。问题在于，当理论表明取样率和搜索行为的极限时，任何执行都使用了有限的群体或样本点集合。基于有限样本点的估算不可避免地会出现取样错误，并有可能使搜索轨迹远远偏离理论预测。这个问题在实际中表现为过早地失去群体的多样性，搜索收敛到次优解。"

Eshelman 和 Schaffer[105]讨论了避免过早收敛的一些策略，包括（1）配对策略(mating strategy)，又称为近亲预防(incest pevention)①；（2）使用均匀杂交(uniform crossover)，见4.6节；（3）检测群体中的重复串，类似于拥挤模型(crowding model)，见第4.1节。

在这一领域的多数工作都是关于：

- 研究由取样机制引起的误差的大小和种类的，
- 研究函数本身的特征的。

这两方面问题是紧密相关的，我们将在4.1和4.2节中依次讨论它们：4.3节将说明一类称之为收缩映射遗传算法的收敛结果，它是基于Banach不动点理论的；4.4节是变群体规模遗传算法的实算结果；4.5节简单讨论几个处理约束的方法；最后一节说明增强遗传搜索的一些其他思想。

4.1 取样机制

在遗传搜索的演化过程中有两个重点：群体的多样性和选择性压力(population diversity, selective pressure)。这两个因素是紧密相关的：增加选择性压力就会降低群体的

---

① 有关近亲预防的技术在货郎担问题（TSP）中的应用，见第10章。

多样性，反之亦然。换句话说，强的选择性压力导致遗传搜索过早收敛；弱的选择性压力使搜索毫无效率。因此，保持这两个因素的平衡是很重要的；取样机制就是试图达到这一目标。正如 Whitley[395]所评述的：

> “可以说，在遗传搜索中有两个可能是仅有的主要因素：群体多样性和选择性压力……从某种意义上说这只是 Holland 及其他人有关‘探寻’对‘开发’(exploration versus exploitation)思想的变种。许多其他用来协调遗传搜索的参数其实都间接地影响了选择性压力和群体多样性。当选择性压力增加，搜索就会集中于群体中的顶层个体，但是这种‘开发’会损失群体多样性。降低选择性压力或使用大群体会增加搜索的‘探寻’，因为有更多的基因型和更多的模式包含在搜索中。”

可能公认最早的工作是 De Jong[82]在 1975 年做的。他认真考虑了前面章节所提到的几个样本选择变量。第一个变量，又称精华模型(elitist model)，强制保存最好染色体。第二个变量，期待值模型(expected value model)，减少了选择规则的随机误差。这是通过对每个染色体 v 引入一个计数来实现的，此计数初始设定为 $f(v)/\overline{f}$，且当染色体被选择杂交或变异时就相应地减去 0.5 或 1。当染色体计数小于零，染色体就不再被选择。第三个变量，精华期待值模型(elitist expected value model)，为前面两个变量的组合。第四个模型拥挤因子模型(crowding factor model)是用一个新产生的染色体替换一个与新染色体相像的旧染色体。

1981 年，Brindle[49]考虑了一些更进一步的改进：*确定取样*(deterministic sampling)、*无退还随机取样*(remainder stochastic samping without replacement)、*随机竞赛*(stochastic tournament)及*有退还余数随机取样*(remainder stochastic sampling with replacement)。这些研究肯定了这样的改进比简单选择优越。特别是*有退还余数随机取样法*，它根据每个染色体在新群体中出现的预测值的整数部分分配样本数，按照小数部分来竞争群体中的其他位置，这是最成功的一个例子，并被许多研究者作为标准而采纳。1987 年，Baker[23]通过使用一些精确定义的方法对这些改进进行了复杂的理论研究，同时提出了一个新的改进版本，称为*随机普遍取样*(stochastic universal sampling)。此方法使用了一个单轮盘，该轮盘按照第 2 章的标准方法构造，轮盘上刻有数目等于群体规模的等距标度，每个标度相对于一个个体。

另一个方法是基于引入人工加权(artificial weights)：染色体根据它们的等级(rank)而不是实际的评价值按比例地被选择（见[22]、[391]）。这些方法都是基于一种认识：快速地过早收敛的主要原因是*超级*个体的存在，这些超级个体的适应值要比群体的平均适应值大得多，以致于它们有大量的后代，且由于群体规模是固定的，它们就会阻止其他个体的子代在下一代中出现。经过不多的代数，一个超级个体可能会排除有希望的染色体材质，并造成快速收敛到可能是局部的最优解。

有许多方法根据等级来确定后代的数量。如 Baker[22]使用用户定义的值 MAX 作为预测子代数目的上限，并做一条通过 MAX 的直线，这样直线下的面积等于群体规模。用这种方法，可以很容易地确定预测子代数及“相邻”个体的差别。如设 MAX = 2.0 及 *pop_size*

$=50$，则“相邻”个体和预测子代数目之间的差别为 0.04。

另一种方法是使用用户定义的参数 q，并定义一个线性方程，如

$$prob\,(rank) = q - (rank-1)\,r$$

或者一个非线性方程，如

$$prob\,(rank) = q\,(1-q)^{rank-1}$$

两个方程都返回一个以位置等级来分级的被选择概率（等级= 1 的意味着为最好个体。等级= *pop_size* 意味着为最差个体）。

这两种方案都允许用户影响算法的选择性压力。对线性函数要求

$$\sum_{i=1}^{pop_size} prob(i) = 1$$

这意味着 $q = r\,(pop_size - 1)\,/\,2 + 1/\,pop_size$

如果 $r = 0$ ($q = 1/\,pop_size$)，那么就根本没有选择性压力：所有的个体都有相同的选择概率。如果 $q - (pop_size - 1)\,r = 0$，那么

$$r = 2/\,(n\,(n-1)),\ \text{且}\ q = 2/\,n$$

则提供最大选择性压力。换句话说，如果一个线性方程被选择计算分级个体概率，单参数 q，在 $1/\,pop_size$ 和 $2/\,pop_size$ 之间的值将能控制算法的选择性压力。例如，如果 $pop_size = 100$，并且 $q = 0.015$，那么 $r = q\,/\,(pop_size - 1) = 0.00015151515$ 且 $prob\,(1) = 0.015$，$prob\,(2) = 0.0148484848, \cdots\cdots, prob\,(100) = 0.00000000000000000051$。

对一个非线性方程，参数 $q \in (0, 1)$与群体规模无关；大的 q 意味着算法较强的选择性压力。例如，如果 q=0.1 且 $pop_size = 100$，那么 $prob(1) = 0.100$，$prob(2) = 0.1 \times 0.9 = 0.090$、

$prob(3) = 0.1 \times 0.9 \times 0.9 = 0.081, \cdots\cdots, prob(100) = 0.000003$。注意：

$$\sum_{i=1}^{pop_size} prob(i) = \sum_{i=1}^{pop_size} q(1-q)^{i-1} \approx 1\ ^{①}$$

这些方法虽然在某种情况下改进了遗传算法，但仍然存在明显的缺陷。首先，它们把决定何时使用这些机制的任务交给了用户。其次，它们忽略了不同染色体的相对评价信息。第三，它们对所有的情况都等同处理，而不顾问题的大小。最后，基于分级加权的选择步骤违反了模式定理。另外，正如一些研究所展示的[23,391]，它们阻止了缩放问题（将在下一部分讨论）。当然，它们较好地控制了选择性压力，而且，提供一次一个地复制，从而使搜索更集中。

另外，一个可供选择的方法是竞赛选择(tournament selection)[118]，它组合了许多有趣而有效的方法。这种方法在一个单迭代里选择数量为 k 的个体，并从这 k 个个体中选择最好个体到下一代中。这一过程重复 *pop_size* 次。很明显，k 值增大就增加了这一过程的选择性压力；许多应用所使用的经验值为 $k = 2$，即所谓的竞赛度(tournament size)。考虑到 Boltzmann 选择，可以加入模拟退火的一些成分。Boltzmann 选择是这样的：两个成员

---

① 由 = 换成 ≈ 的步骤很容易：完整的表达为 $prob(i) = c \cdot q(1-q)^{i-1}$，其中 $c = 1/\,[1-(1-q)^{pop_size}]$。

i 和 j，彼此相互竞争，通过下面公式决定胜者

$$\frac{1}{1+e^{\frac{f(i)-f(j)}{T}}}$$

其中 T 为温度，$f(i)$和$f(j)$分别为成员 i 和 j 的目标函数值（表达为求最小问题）。

在[16]中，Bäck 和 Hoffmeister 讨论了各种选择方案。他们把选择方案划分成动态的和静态的方法——静态选择要求选择概率在代之间保持不变，如分级加权选择；而动态选择则是可变的，如比例选择。另一种方法将选择方案划分成灭绝法(extinctive)和保存法(preservative) —— 保存选择要求对每个个体都有一个非零的选择概率；而灭绝选择则没有这样的要求。灭绝选择可进一步划分成左选择和右选择：在左灭绝选择里，最好个体被禁止进行复制以避免超级个体造成的过早收敛，而右灭绝选择没有这样的限制。另外，一些选择方案是从纯选择的意义上说的：父代只允许在一代里进行繁殖，即，不论其适应值大小，每个个体的寿命仅限于一代。在第 8 章，在我们讨论演化策略并将它们与遗传算法进行比较时，还会回到灭绝选择、纯选择。一些选择是本着代的意义：父代是固定的，直到下一代中所有的子代都产生为止；在即时选择时，一个子代立即替换其父代。一些选择是精华选择，其含义是一些或所有父代被允许和它们的子代一起演化选择 ——我们已经在精华模型[82]中看到这样的选择。

在本书讨论的多数实例中，我们使用了一个新的基本选择算法的两步变种。但是，这种改进不只是一个新的选择机制；它能使用任何目前所开发的取样方法，而且它的设计可以降低一些函数的特征性所带来的不应有的影响。当然，它仍然归属于动态、保存、代及精华选择这一类。

```
        procedure modGA
begin
        t ← 0
        initialize P(t)
        evaluate P(t)
        while (not termination-condition) do
        begin
                t ← t+1
                select-parents from P(t-1)
                select-dead from P(t-1)
                form P(t): reproduce the parents
                evaluate P(t)
        end
end
```

图 4.1 modGA 算法

这一改进的遗传算法(modGA)的结构如图 4.1 所示。这些涉及经典遗传算法的改进是：在 modGA 中，我们没有执行“select $P(t)$ from $P(t-1)$”这一选择步骤，而是独立选择 r 个并不一定不同的染色体进行复制并让 r 个不同的染色体死亡。这些选择用到了串的相对适应值：一个执行结果高于平均的串有更多的机会被选择复制；低于平均的串有更多的机会死亡。modGA 的“select-parents”和“select-dead”步骤执行后，在群体中有三组不一定是完全区分的串：

• r 个进行复制的（有可能相同的）串

- r 个死亡的串
- 剩下的串（称为中性串）

一代中，中性串的数目至少是 $pop_size-2r$，至多是 pop_size-r；其值依赖于被选择的不同父代的数及在“父代”和“死亡”名录中重叠串的数。然后就形成了一个大小为 $P(t+1)$ 的新群体，即 pop_size-r 个被选择死亡以外的串和 r 个父代所产生的 r 个子代。

目前，此算法有一个步骤还存在潜在的问题：如何选择 r 个染色体死亡。很明显，我们希望该选择使较好的染色体有较小的死亡机会。这可以通过改变构造新的群体 $P(t+1)$ 的方法来实现：

[步骤 1] 从 $P(t)$ 中选择 r 个父代。每一个被选择的染色体（或者说每一个被选择的染色体副本）当被精确地用于一个遗传算子时，被标记一次。

[步骤 2] 从 $P(t)$ 中选择 pop_size-r 个不同的染色体，并将它们复制到 $P(t+1)$ 中。

[步骤 3] 让 r 个父代染色体精确地繁殖出 r 个后代。

[步骤 4] 将这 r 个新子代插入到群体 $P(t+1)$ 中。

上述选择步骤 1 和步骤 2 都是根据染色体的适应值进行的（随机均匀取样法）。

上述选择方法及前面的选择方法并没有本质的不同。首先，父代和子代都有较好的机会在新代中出现：一个平均之上的个体有较好的机会被选择作为一个亲体（第 1 步），同时也有较好的机会被选择进入有 pop_size-r 成员的新群体中（第 2 步）。这样，其后代中的一个或多个将取代剩余 r 个位置。其次，我们应用遗传算子在所有的个体上，而不是经典变异中的个体位上。这将对所有用在演化程序中的算子进行均匀的处理（一个演化程序 GENOCOP 用到了几个遗传算子，见第 7 章）。所以，如果用到三个算子（即变异、杂交，反转），那么父代中的一部分将进行变异，一些进行杂交，其余的进行反转。

改进的 modGA 法和经典的遗传算法有一些相似的理论特性。我们可以重写第 3 章的生长公式(3.3)：

$$\xi(S,t+1)\geq\xi(S,t)\cdot p_s(S)\cdot p_g(S) \tag{4.1}$$

这里，$p_s(S)$ 表示模式 S 的存活概率；$p_g(S)$ 表示模式 S 的生长概率。模式 S 的生长发生在几个平均之上的模式副本被拷贝到新群体中的选择阶段（生长阶段）。模式 S 的生长概率 $p_g(S)=eval(S,t)/\overline{F(t)}$，对平均之上的模式，$p_g(S)>1$。然后，被选择染色体经过遗传算子杂交和变异（收缩阶段）后必须是存活的。正如第 3 章所讨论的，模式 S 的存活概率 $p_s(S)$ 为：

$$p_s(S)=1-p_c\frac{\delta(S)}{m-1}-p_m\cdot o(S)<1$$

公式 4.1 说明对短的、低阶模式，$p_g(S)\cdot p_s(S)>1$；正是由于这一点，一些模式在后续代中接受几何增长的副本数。对算法的修正版本 modGA 也同样具有这一特性。在 modGA 算法中，模式 S 预计的染色体数同样是旧群体中染色体数 $\xi(S,t)$、存活概率（$p_s(S)<1$）和生长概率 $p_g(S)$的乘积——所不同的仅是对生长和收缩阶段的解释及其相对

次序。在 modGA 版本中，收缩阶段排第一：$n-r$ 个染色体被选择进入新群体。存活概率被定义为模式 S 中未被选择死亡的染色体的分数。生长阶段其次，它将加入 r 个新后代。模式 S 的生长概率 $p_g(S)$为模式 S 被 r 个亲体产生的新后代所扩容的概率。对短的、低阶模式，$p_s(S) \cdot p_g(S)>1$，且该模式在后续代中接受几何增长的性能实算。

modGA 算法的思想之一是更好地利用可用的存储资源：群体的规模。新的算法避免了在新群体中同样染色体的过多副本，尽管它可能还会偶然发生，但与其他方法相比已有很大不同了。而经典的算法却很容易产生这种多副本。而且，多次出现的超级个体很可能产生连锁反应：不断在下一代中产生大量的完整拷贝。这样造成有限的群体规模里独立染色体的减少，低的空间利用率降低了算法的执行效果；注意，遗传算法的理论基础是假定群体规模是无限大的。

在 modGA 算法里，我们可能会碰到染色体的一定数量的家庭成员，但所有这些成员都是不同的（这里一个家庭指的是后代和父代）。

例如，一个染色体预计在 $P(t+1)$中出现次数 $p=3$。假定经典的遗传算法使用常用的方案，杂交概率 $p_c=0.3$ ，变异概率 $p_m=0.003$。进行选择后，该染色体在复制前在 $P(t+1)$ 代中有 $p=3$ 个完整拷贝数。经过复制后，假定染色体长度 $m=20$，则该染色体预计在 $P(t+1)$代中的完整拷贝数变为：$p\cdot(1-p_c-p_m\cdot m)=1.92$。因此可以说，减去不同染色体数后，下一代中将有两个这样染色体的完整副本。

modGA 中用到的改进是基于拥挤因子模型的思想[82]：一个新产生的染色体替换某个旧的。所不同的是在拥挤因子模型里，死亡染色体是从与新染色体相似的那些染色体中选择的，而在 modGA 中，死亡染色体是那些低适应值的染色体。

对小的参数值 r，modGA 法属于稳态遗传算法(SSGA)[394,382] 一类；GAs 和 SSGAs 的主要不同在于每代中后者只有几个个体被改变。modGA 和分级系统（第 12 章）(Classifier system) 有些相似之处：分级系统的遗传组份尽可能少地改变群体。在 modGA 中，我们可以通过参数 r（它决定参加复制的染色体数及死亡的染色体数）调整这种改变。在 modGA 中，pop_size-r 个染色体被毫无变化地放到新群体中。对 $r=1$，每代中只有一个染色体被替换。最近，*Mühlenbein* [289]建议了繁殖遗传算法(Breeder GAs, BGA)，它选择 r 个最好个体，并随机配对直到子代数等于群体规模。子代替换父代且当前发现的最好个体保留在群体中。

4.2 函 数 特 征

modGA 算法提供了一种从旧群体中构造新群体的新方法。看起来似乎一些其他方法在解决与优化函数的特征性有关的问题时可能是有用的。通过几年的研究，我们看到了三个基本方向。一个是借用了变更系统熵的模拟退火技术，见[359]，其中作者通过使用全局温度参数的热力学算子控制群体收敛率。

另一个方向是按照等级权值分配生殖个数，而不是在前面章节中讨论的按照实际评价值，因为分级加权自动引入一个对群体归一化的比例。

最后一个方向是通过引入比例机制集中于固定函数本身。按照 Goldberg[154]的分类，我们将这样一些机制划分成三类：

1．线性比例法

在这种方法里，实际染色体的适应值缩放成

$$f_i' = a * f_i + b$$

参数 a，b 的选择通常是使平均适应值被其本身映射，而且最好个体被加上一个想要的对应于平均适应值的一个倍数。这种机制尽管很强大，但可能引入必须被处理的负评价值。另外，参数 a、b 和群体的寿命有关，而与问题无关。

2．σ截断法

这种方法不仅改进了线性比例中评价值为负的问题，而且并入了与问题有关的信息到其本身的映射里。新适应值是这样计算的：

$$f_i' = f_i + (\overline{f} - c * \sigma)$$

这里 c 通常为一个 1～5 之间的小整数，σ为群体的标准偏差；对可能出现的负值，f'设定为零。

3．幂法则比例法

在这种方法里，初始的适应值被加入特定的幂：

$$f_i' = f_i^k$$

某些 k 可能接近于 1。参数 k 缩放了函数 f，但[142]的研究表明，k 的选择应与问题有关，在同一研究中，作者用 $k = 1.005$ 获得了一些实算的改进。

和函数特征性有关的最值得注意的问题是相对适应值的不同，以下面两个函数为例：$f_1(x)$和 $f_2(x) = f_1(x) + const$。因为它们样式基本相同，即它们有同样的最优值，可以预测它们在优化时有相同的难度。但是，如果 $const \gg \overline{f_1}(x)$，那么函数 $f_2(x)$将比函数 $f_1(x)$ 收敛得慢。实际上，在极端的情况下，第二个函数将用一个全随机搜索来优化；这种搜索在群体的早期还是可以忍受的，但在后期却不能。相反，$f_1(x)$可能收敛得太快，而使算法收敛到局部最优。

另外，由于群体规模是固定的，遗传算法的行为在每次运行时是不一样的，这是由于有限样本所造成的误差。考虑函数 $f_3(x)$，其中一个样本 $x_i^t \in P(t)$接近局部最优且 $f(x_i^t)$ 远大于平均适应值 $\overline{f}(x^t)$，即 x_i^t 是一个超级个体。更进一步假定不存在靠近全局最优的点 x_j^t。这可能是由高度不平滑函数造成的。在这种情况下，搜索很快收敛到局部最优。因此，群体 $P(t+1)$ 将造成靠近局部最优点成员的过饱和。从而降低了进行全局搜索的机会。这种行为在演化阶段的后期是允许的，甚至在最末阶段也是合理的，但在早期，却令人十分烦恼。而且，正常的后期群体在算法的后阶段被相似适应值的染色体所饱和，这些染色体与配对过程有密切的关系。因此，用传统的选择技术，复制实际上是随机的。这种行为正好和想要的相反，对初始阶段的群体，选择过程中染色体的相对适应值相互

影响越小越好，而在后阶段，越大越好。

一个知名的系统，GENESIS 1.2ucsd，使用了两个参数来控制被优化函数特征性的搜索：比例窗(scaling window)和σ截断因子(sigma truncationfactir)。该系统是求一个函数的最小值：在这种情况下，通常返回的评价函数 *eval* 为：

$$eval(x)=F-f(x)$$

这里 F 为一个常数，且对所有 $x, F>f(x)$。正如前面所讨论的，对 F 挑选较差将引起搜索不好的效果，而且 F 可能事先并不知道。GENESIS 1.2ucsd 中的比例窗 W 允许用户控制常数 F 如何被更新：如果 $W>0$，则系统设定 F 为最近 W 代中出现的最大 $f(x)$值。$W=0$ 表明一个无穷大的比例窗，即 $F=\max\{f(x)\}$。如果 $W<0$，则用户可以使用前面所讨论的另一种方法：σ截断。

应该指出的要点是在算法中使用的终止条件（termination condition）。最简单的终止条件是检查当前代数：如果总的代数超过预定的常数则搜索终止。在引言中的图 0.1 里，终止条件是表达为：对某一常数 T，“$t\geq T$ ”。在许多演化程序的版本里，并不是所有的个体都需要再评价：它们中的一些毫无变化地从一代传到另一代。在这种情况下，与一些其他传统的算法相比，计数函数评价的次数是很有意义的，通常该数与演化代数成比例，当函数评价次数超过预定常数时，中断搜索。

当然，上述中断条件假设用户知道影响搜索长短针对函数特征性的知识。在许多实例中，当取得重大改进的机会相当渺茫时，断定总代数或函数评价代数多少是很难的。

有两类基本的中断条件用到了搜索的特征性以作出中断的决定。一类是基于染色体结构（基因型）的；另一类是本着特殊染色体（表现型）意义的。第一类中断条件通过检查收敛的等位基因来检测群体的收敛性，这里如果一些预定群体中该等位基因的百分数一样（或二进制表达相似），则等位基因被看成收敛。如果收敛的等位基因数超过全部等位基因数的某个百分比，搜索中断。第二类中断条件用一个度量预定代数算法的进度：如果进度小于该方法中的一个参数ε，则算法中断搜索。

4.3 收缩映射遗传算法

遗传算法的收敛性在演化计算领域中是一个很重要的极具挑战性的理论问题。一些研究者从不同的方面研究了这个问题。Goldberg 和 Segrest[163]提供了遗传算法的有限马尔科夫链分析（有限的群体只有复制和变异）。Davis 和 Principe[80]研究了将现存的模拟退火算法理论基础外推到遗传算法模型的马尔科夫链的可能性。Eiben，Aarts 和 Van Hee[96]建议了一个精炼的遗传算法，它将遗传算法和模拟退火统一了起来：讨论了针对该精炼遗传算法的马尔科夫链分析，并给出了演化过程找到最优解的概率为 1 的条件。Kingdon 研究了开始点、收敛性及遗传算法难解问题。总结了竞争模式的概念，并给出了这种模式的收敛概率。几个研究者考虑了欺骗问题的各种定义[154]。最近，Rudolph[334]证明经典的遗传算法永远不会收敛到全局最优，但修订版（即精华模型）可以在群体中保存最好解。

一个解释遗传算法收敛特性可能的方法是基于 Banach 不动点理论[386]。它提供了非精华模型的遗传算法收敛性的直觉性的解释；仅有的要求是在后续群体中必须有改进，并不一定是最好个体的改进。Banach 用不动点理论处理了矩阵空间的收缩映射问题。它证明这样的映射 f 有唯一的不动点，即仅有一个 x 使 $f(x)=x$。不动点技术被作为定义计算语义学的强大工具而被广泛接受。例如，通常给出一个程序或计算的表述语义学，来作为定义在一个适当的完整网格上的连续映射的最低限的不动点。但是，不像传统意义上的语义学，我们发现这些度量空间提供了一个非常简单而且自然的表达遗传算法语义学的方法。遗传算法能被定义成群体间的变形。假定现在我们能够发现这样一个计量空间，其中的变换都是收缩性的。在此情况下，给出的遗传算法的语义学就当作基础变换的不动点。因为任意这样的变换都有唯一不动点，得到的遗传算法的收敛性就成为一个简单的必然结果。

直觉地说，一个计量空间是一个集合和一个函数组成的一个有序的对，它允许我们度量集合中任何成员对的距离。如果 $f(x)$ 和 $f(y)$ 之间的距离小于 x 和 y 之间的距离[①]，则定义在这样的一个集合映射 f 是收缩性的。

现在，让我们更正规地定义这些基本概念。用 R 表示实数集合。如果对任何元素 $x, y \in S$ 满足下面条件，则集合 S 和映射 $\delta: S \times S \rightarrow R$ 就构成了度量空间。

- $\delta(x, y) \geq 0$，当且仅当 $x=y$ 时 $\delta(x, y)=0$
- $\delta(x, y)=\delta(y, x)$
- $\delta(x, y)+\delta(y, z) \geq \delta(x, z)$

映射 δ 被称为距离。我们用 $\langle S, \delta\rangle$ 表示度量空间。

设 $\langle S, \delta\rangle$ 为一个度量空间，$f: S \rightarrow S$ 为一个映射。当且仅当常数 $\varepsilon \in[0,1)$ 并对所有的 x, $y \in S$

$$\delta(f(x), f(y)) \leq \varepsilon * \delta(x, y)$$

都成立，我们就说 f 是收缩的。

为了公式化 Banach 理论，我们必须定义度量空间完备性的概念。如果对所有 $\varepsilon>0$，存在一个 k，当 $m, n>k$ 时，$\delta(p_m, p_n)<\varepsilon$，我们说度量空间 $\langle S, \delta\rangle$ 的元素序列 $p_0, p_1, \ldots$ 是柯西序列。如果任意柯西序列 $p_0, p_1, \ldots$ 都有一个极限 $p=\lim\limits_{n \rightarrow \infty} p_n$，我们说度量空间是完备的。

我们现在准备用公式表达 Banach 定理。定理的证明首次由[25]完成，并可在大多数拓扑学手册中查到（见[94]，p60）。

定理 Banach 不动点理论（Banach fixpoint theorem[25]）设 $\langle S, \delta\rangle$ 是一个完全度量空间，$f: S \rightarrow S$ 是一个收缩映射。则 f 有一个唯一的不动点 $x \in S$，使得对任意 $x_0 \in S$，

$$x=\lim_{i \rightarrow \infty} f^i(x_0)$$

① 实际上，正如我们以后会看到的，小于的意义要比一般上的意义更强。

其中 $f^0(x_0) = x_0$，$f^{i+1}(x_0) = f[f^i(x_0)]$。

Banach 定理可很自然地应用于遗传算法，即如果我们构造度量空间 S 使其成员成为群体，那么任何收缩映射 f 都有唯一的不动点；根据 Banach 定理，该不动点是通过 f 应用于一个任选的初始群体 $P(0)$的迭代而获得的。这样，如果我们发现一个合适的度量空间使在此空间里的遗传算法是收缩的，那么我们就可以在不动点上获得算法的收敛性，而与初始群体的选择无关。应当说明这样的构造对一个轻度修正的遗传算法是可能的，称之为收缩映射遗传算法(Contractive Mapping Genetic Algorithms, CM-GA)。

不失一般性，假定我们处理求最大问题，即如果 $eval(\bar{x}_i) > eval(\bar{x}_j)$，则解 $\bar{x}_i$ 要比解 $\bar{x}_j$ 好。

假定，群体规模 $pop_size = n$ 是固定的；每个群体由 n 个个体组成，即 $P = \{\bar{x}_1, ..., \bar{x}_n\}$。让我们考虑一个对群体 P 的评价函数 $Eval$；假定为：

$$Eval(P) = \frac{1}{n}\sum_{\bar{x}_i \in P} eval(\bar{x}_i)$$

这里 $eval$ 返回的是群体 P 中一个个体 $\bar{x}_i$ 的适应值。集合 S 是由所有可能的群体 P 组成，即任何向量 $\{\bar{x}_1, ..., \bar{x}_n\} \in S$。

现在，我们在度量空间$\langle S, \delta\rangle$定义一个映射距离 $\delta: S \times S \to R$ 及一个收缩映射 $f: S \to S$。群体在度量空间 S 里的距离δ可以定义为：

$$\delta(P_1, P_2) = \begin{cases} 0 & \text{若 } P_1 = P_2 \\ |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)| & \text{若 } P_1 \neq P_2 \end{cases}$$

这里 M 为域里的函数 $eval$ 的上限，即对所有个体 $\bar{x}$，$eval(\bar{x}) \le M$。因此，对所有可能群体 P，$Eval(P) \le M$。可以得出，

- 对所有群体 P_1 和 P_2，$\delta(P_1, P_2) \ge 0$； 当且仅当 $P_1 = P_2$，$\delta(P_1, P_2) = 0$；
- $\delta(P_1, P_2) = \delta(P_2, P_1)$；
- $\delta(P_1, P_2) + \delta(P_2, P_3) = |1+M-Eval(P_1)| + |1+M-Eval(P_2)| + |1+M-Eval(P_2)| + |1+M-Eval(P_3)| \ge |1+M-Eval(P_1)| + |1+M-Eval(P_3)| = \delta(P_1, P_3)$。

因此 $\langle S, \delta\rangle$ 为一个度量空间。

而且，度量空间$\langle S, \delta\rangle$是完备的。因为对任何群体中的柯西序列 $P_1, P_2, \ldots$，存在 k，对所有 $n>k$，$P_n = P_k$。这就意味着对所有柯西序列 P_i（$i \to \infty$)都有一个极限[①]。

现在，我们准备讨论收缩映射 $f: S \to S$，该映射为遗传算法运行中的一个简单的单迭代[②]（见图 4.2），假定群体 $P(t)$到群体 $P(t+1)$有改进（以函数 $Eval$ 表达）。在这种情况下，$f(P(t)) = P(t+1)$。换句话说，如果 $Eval(P(t)) < Eval(P(t+1))$，遗传算法的第 t 个迭代将作为一个收缩映射算子 f。如果没有改进，我们就不对该迭代计数，即再次运行选择和重组过程。

这种修正的遗传算法——收缩映射遗传算法的结构(CM-GA)如图 4.2 所示。

① 注意对遗传算法，我们主要处理有限度量空间，因为对所有可能群体都只有有限数量的成员；这样 Banach 所要求的度量空间的完备性在这种情况下总是满足的。在上述情况下，对任意集合 S，$\langle S, \delta\rangle$ 都是完备的。

② 一个运行，我们这里指任何可观察的计算序列。

```
procedure CM-GA
begin
      t = 0
      initialize P(t)
      evaluate P(t)
      while (not termination-condition) do
      begin constractive mapping   f(P(t)) → P(t+1)
            t = t+1
            select P(t) from P(t−1)
            recombine P(t)
            evaluate P(t)
            if   Eval(P(t−1)) ≥ Eval(P(t))
            then     t = t−1
      end
end
```

图 4.2　收缩映射遗传算法

CM-GA 修正的迭代确实能满足收缩映射的要求。很明显，如果迭代 $f: P(t) \to P(t+1)$ 改进了群体的函数值 *Eval*，即如果

$$Eval(P_1(t)) < Eval(f(P_1(t))) = Eval(P_1(t+1)),$$

并且 $$Eval(P_2(t)) < Eval(f(P_2(t))) = Eval(P_2(t+1))$$

那么 $\delta(f(P_1(t)), f(P_2(t))) = |1+M-Eval(f(P_1(t)))| + |1+M-Eval(f(P_2(t)))|$

$$< |1+M-Eval(P_1(t))| + |1+M-Eval(P_2(t))| = \delta(P_1(t), P_2(t))$$

不断处理算法的特定实现过程，改进将不小于给定的最小实数。

总之，CM-GA 满足 Banach 不动点定理的假设：群体空间 $\langle S, \delta \rangle$ 是一个完备的度量空间，改进了群体的评价函数 *Eval* 的循环 $f: P(t) \to P(t+1)$ 是收缩的。因此，

$$P^* = \lim_{i \to \infty} f^i(P(0))$$

即，CM-GA 算法收敛到群体 P^*，它是所有群体空间里的唯一不动点。

很明显，P^* 代表产生全局最优的群体。注意 *Eval* 函数被定义为：

$$Eval(P) = \frac{1}{n} \sum_{\bar{x}_i \in P} eval(\bar{x}_i)$$

这就意味着当该群体中所有个体都有相同的值（全局最大）时，即得到不动点 P^*。而且，P^* 与初始群体 $P(0)$ 无关。

一个有趣的问题是当评价函数 *eval* 有不止一个最大值的情况。在这种情况下，对于最优群体 P_1, P_2，收缩映射遗传算法不是真正意义上的收缩映射。

$$\delta(f(P_1), f(P_2)) = \delta(P_1, P_2)$$

另一方面，在这种情况下，CM-GA 收敛到一个可能的最优群体。实际上，算法的每次运行都收敛到一个最优群体。

初看结果是很令人惊讶的：对收缩映射遗传算法，初始群体的选择只是影响收敛速度。基于 Banach 定理的收缩映射遗传算法的意图是在无限的时间里收敛到全局最优。但很有可能在算法的某个阶段，长时间没有可以接受的新群体出现，算法不断循环试图找到新群体 $P(t)$。换句话说，变异和杂交算子应用于一个特殊的次优群体而不能产生“更好”

群体，且算法的循环又试图进入下一个收敛阶段。群体间距离δ和评价函数 *Eval* 的选择一方面要尽可能的简单，另一方面这种选择又会影响收敛速度，而且与应用相关。

4.4 变群体规模的遗传算法

群体规模是所有遗传算法的使用者所必须考虑的一个重要因素，在许多应用中甚至可能是至关重要的。如果群体规模太小，遗传算法可能收敛得太快；如果太大，遗传算法可能浪费计算资源：为一个改进所耗费的等待时间太长。正如我们先前讨论的（4.1 节），遗传算法的演化过程有两个关键点：群体多样性和选择性压力。很明显，这两个因素都受群体规模的影响。

几个研究者从几个方面研究了遗传算法群体规模。Grefenstette[160]用了一个 meta-GA 来控制另一个遗传算法的参数(包括群体规模和选择方法)。Goldberg[151,153]对最优的群体规模进行了理论分析。[343]是一项对遗传算法控制参数的影响的研究（函数优化在线性能）。另外，在[206]和[59]中有关于群体规模的实算报告。最近 Smith[362]建议了一个根据选择误差调节群体规模的算法。

在这一节，我们将讨论变群体规模遗传算法(GA with Varying Population Size——GAVaPS)[12]。该算法没有使用前面所考虑的各种选择机制（4.1 节），而是引入了染色体“代龄”的概念，它等同于染色体“存活”的代数。这样，染色体的代龄替代了选择的概念，因为它依赖于个体的适应值，并在过程的每个阶段影响群体规模。看起来这种方法比先前考虑的选择机制更“自然”：毕竟，增龄过程在自然界是众所周知的。

建议的变群体规模的方法与演化策略和其他一些子代和父代竞争生存的方法的部分内容相似（第 8 章）。但有一个重要的不同：在其他方法里群体规模保持不变，而在 GAVaPS 里群体规模是随时间变化的。

本工作的另一个动机是基于下面的观察：几位研究者检查了在遗传算法中引入遗传算子的适应概率的可能性[77,115,343,367,368]；其他技术，如演化策略[349]早已引入算子的适应概率（我们将在 4.6 节和第 8 章进行简要的讨论）。看起来似乎有理由假定在演化过程的不同阶段，不同的算子有不同的重要性，系统应该对它们的频率和范围进行自我调节。对群体规模也同样应该调节：在演化过程的不同阶段，不相同的群体规模可能是“最优的”，因此用一些启发式规则进行调节搜索当前阶段的群体规模的实算是很有必要的。

GAVaPS 算法在时刻 t 处理染色体群体 $P(t)$。在“重组 $P(t)$”阶段，一个新的辅助群体产生，即子代群体。辅助群体规模正比于原始的群体规模；辅助群体含有 $AuxPopSize(t) = \lfloor PopSize(t) * \rho \rfloor$ 个染色体，参数ρ为复制率。群体中的每个染色体都以与其适应值无关的相等的概率被选择复制，即将其子代放入辅助群体里。后代是由遗传算子（杂交和变异）选择染色体而产生的。因为染色体的选择过程与它们的适应值无关，即不存在与适应值有关的选择过程，我们引入染色体代龄的概念和寿命参数。

GAVaPS 的结构如图 4.3 所示。

```
procedure GAVaPS
begin
    t = 0
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
    begin
        t = t+1
        increase the age of each individals by 1
        recombine P(t)
        evaluate P(t)
        remove from P(t) all individals with age greater than their lifetime
    end
end
```

图 4.3　GAVaPS 算法

寿命参数在演化阶段为每个染色体分配一次（要么所有染色体初始化后对所有染色体，要么重组阶段后对辅助群体成员），对一个给定染色体，此参数在演化过程中保持不变，即从染色体的产生到死亡。这就意味着对“旧”染色体，它们的寿命值不需再计算。染色体的代龄（即染色体存活的代数，初始设定为零）超过其寿命值，染色体就死亡。换句话说，染色体的寿命决定了 GAVaPS 的代数，在此期间染色体保存在群体中：超过其寿命，染色体就死亡。这样，经过一个单迭代，群体规模变为：

$$PopSize(t+1) = PopSize(t) + AuxPopSize(t) - D(t)$$

这里 $D(t)$ 为在第 t 代，染色体死亡的个数。

有许多种分配寿命值的方案。很明显，不考虑任何搜索统计结果，分配一个大于 1 的常数将导致群体规模的指数增长。而且，因为不存在像 GAVaPS 的选择机制，也就不存在选择性压力，所以为寿命参数分配一个常数将造成算法执行效率较低。为了引入选择性压力，应当进行一个更深思熟虑的计算。寿命计算方案应该：（1）加强适应值在平均之上的个体，从而限制适应值在平均之下的个体；（2）调节搜索当前阶段的群体规模，特别是阻止群体的对数生长，降低模拟的消耗。对适应个体的加强应该使它的子代在附加群体中得到平均之上的分配。因为每个个体都有相同的概率进行基因重组，个体子代的预计数与它的寿命值成比例（因为寿命决定了在群体中保存的代数）。所以有平均之上适应值的个体应该享有更高的寿命值。在计算寿命时，应该考虑遗传算法的状态。因此，我们用几个参数度量搜索状态：*AvgFit*、*MaxFit* 和 *MinFit* 分别表示当前群体中平均、最大和最小适应值，***AbsFitMax*** 和 ***AbsFitMin*** 代表到目前为止发现的最大和最小适应值。应该注意到，为了节省计算资源，寿命计算应尽可能容易。

考虑上述的观点，我们设计并实算了几个寿命计算方案。对第 i 个个体的寿命参数 *lifetime*[*i*]可以这样确定（如前面章节所示，我们假定为求非负评价函数的最大值问题）：

(1) 比例分配：

$$\min(MinLT + \eta\frac{fitness[i]}{AvgFit}, MaxLT)$$

(2) 线性分配：

$$MinLT + 2\eta \frac{fitness[i] - AbsFitMin}{AbsFitMax - AbsFitMin}$$

(3) 双线性分配：

$$\begin{cases} MinLT + \eta \dfrac{fitness[i] - MinFit}{AvgFit - MinFit} & \text{若 } AvgFit \geq fitness[i] \\ \dfrac{1}{2}(MinLT + MaxLT) + \eta \dfrac{fitness[i] - AvgFit}{MaxFit - AvgFit} & \text{若 } AvgFit < fitness[i] \end{cases}$$

这里 *MaxLT* 和 *MinLT* 分别代表最大和最小可允许的寿命值，这些值被作为 GAVaPS 参数给出，$\eta = \frac{1}{2}(MaxLT - MinLT)$。

第一种方案——比例分配出自赌盘选择的思想：特定个体的寿命值正比于其适应值（在 *MinLT* 和 *MaxLT* 之间）。但是，这种方案有一个严重的缺点——它没有利用有关个体“目标好坏度”的任何信息，这种好坏度可以通过将适应值与到目前为止发现的最好值进行关联估算出。这种观察促成了线性方案。在该方案里，寿命值按照个体对目前最好个体的相对适应值计算出。然而，如果有太多的个体的适应值等于或近似等于最好值，这种方案将导致分配较长的寿命值，这样会增大群体规模。最后的双线性方案试图对上述两种方案进行折衷。它通过利用平均适应值的信息加剧接近最好个体寿命值之间的差距，同时也考虑了当前发现的最大适应值和最小适应值。

我们用 GAVaPS 算法对下面函数进行了测试：

G1: $-x\sin(10\pi x) + 1$　　$-2.0 \leq x \leq 1.0$

G2: $\mathbf{integer}(8x)/8$　　$0.0 \leq x \leq 1.0$

G3: $x \cdot \mathbf{sgn}(x)$　　$-1.0 \leq x \leq 2.0$

G4: $0.5 + \dfrac{\sin^2\sqrt{x^2+y^2} - 0.5}{(1+0.001(x^2+y^2))^2}$　　$-100 \leq x, y \leq 100$

所选函数覆盖了可能优化函数的较宽范围。函数 G1 和 G4 是有许多局部最大的多峰函数。因为没有可利用的梯度信息，函数 G2 不能被任何梯度技术所优化。函数 G3 代表“欺骗问题”[154]。在求函数最大值时，很容易形成两个方向上的生长，但所选的边界只能获得一个最大值。对随机取样以梯度为基础的技术，它可能导致频繁地发现局部最优。

我们将 GAVaPS 的执行结果和 Goldberg[154]的简单遗传算法 SGA 的执行结果进行了比较。其中对问题的编码方法及遗传算子，SGA 和 GAVaPS 完全相同，即一个简单的二进制编码、两个遗传算子：变异和单点杂交。

实算进行了如下假设。初始群体规模为 20。对 SGA，初始群体规模在整个模拟过程中保持不变。复制率ρ设定为 0.4（该参数对 SGA 无意义），变异率设定为 0.015，杂交率设定为 0.65，染色体长度为 20。所有的实算假定最小和最大寿命值不变，*MaxLT*= 7，*MinLT* = 1。

为比较 SGA 和 GAVaPS，我们选择了两个参数：算法的消耗，由 *evalnum* 代表。它是整个过程运行函数评价的平均次数。执行效率 *avgmax* 表示整个运行过程中发现的最大值的平均值。两个算法都用同样的中断条件：如果连续 *conv* = 20 代最好值没有进步，则算法中断。群体随机进行初始化，并独立运行 20 次。然后测定执行效率和消耗并被 20

次平均。在测试单个参数对执行结果和消耗的影响时，除被测试参数外，上述其他参数保持不变。

图 4.4 显示了对函数 G4 用双线性寿命计算的单个 GAVaPS 运行过程的 *PopSize(t)*及群体的平均适应值（其他函数和其他分配寿命值的方案也有相似的观察结果）。*PopSize(t)*曲线的形状看起来很有趣。首先，当适应值相当高时，群体规模增大。这就意味着 GAVaPS 对最优点做了一个广泛的搜索。一旦最优点的邻近点被找到，算法开始收敛，且群体规模减小。但是，搜索仍在改进。当一个更好解有可能出现时，另一个“人口爆炸”就会发生，随之而来的是另一个收敛阶段。GAVaPS 是通过在演化过程的每个阶段选择群体规模进行自我调节的。

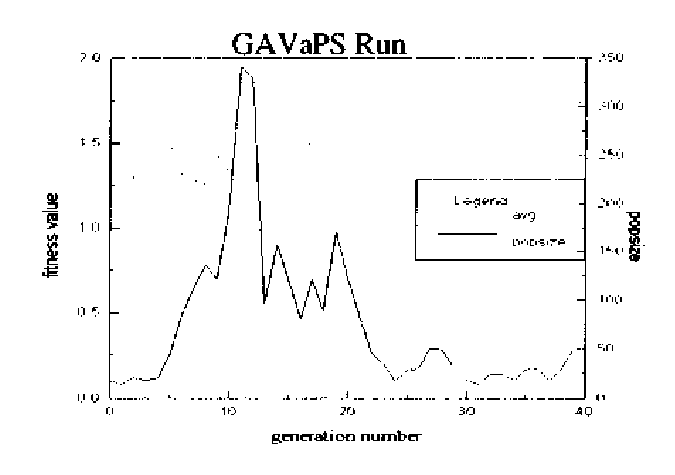


图 4.4 *PopSize(t)* 和 GAVaPS 单个运行时的群体平均适应值

图 4.5～4.6 显示了复制率对 GAVaPS 参数的影响。对 SGA，该值无意义，因为在此情况下，新群体完全覆盖旧群体。对 GAVaPS，该值强烈地影响着模拟的消耗，降低复制率可以减少其值，当然就会损失精度（见有关 *avgmax* 值）。从实算判断，ρ的“最优”选择大约是 0.4。

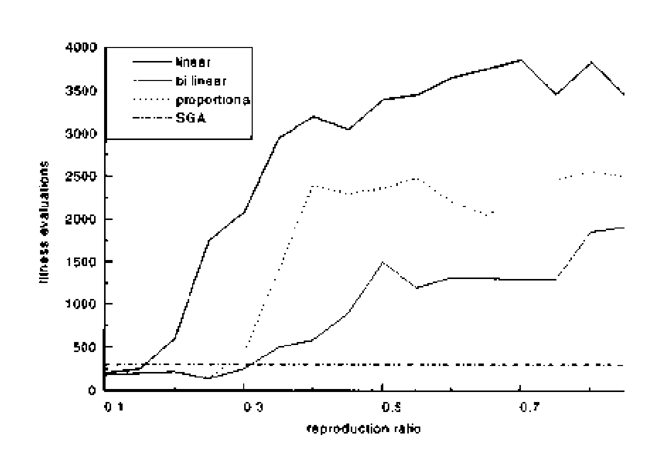


图 4.5 SGA 和 GAVaPS 的比较：复制率对评价数（函数 G4）

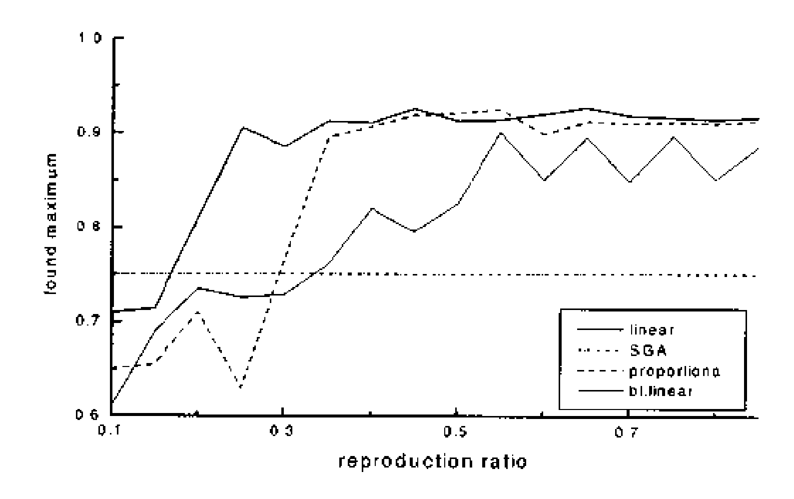


图 4.6 SGA 和 GAVaPS 的比较：复制率对平均执行效果（函数 G4）

图 4.7～4.8 显示了初始群体规模对执行效果 *avgmax* 和算法计算消耗 *evalnum* 的影响。对 SGA，所有运行过程的群体规模都是不变的，等于初始值。正如所预测的那样，

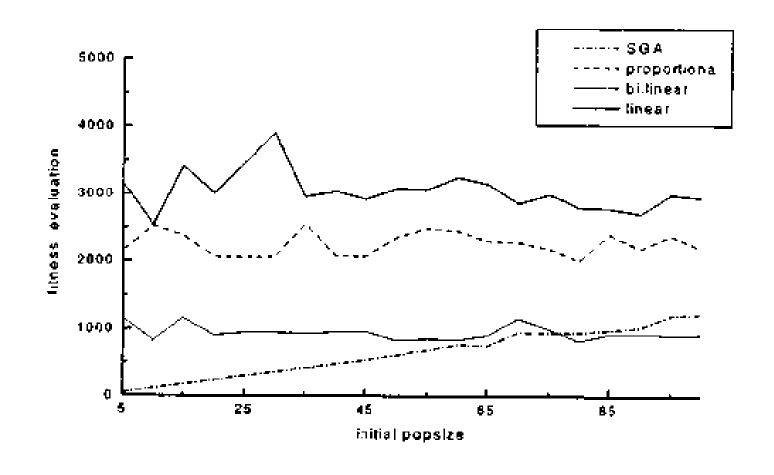


图 4.7 SGA 和 GAVaPS 的比较：初始群体规模对评价数（函数 G4）

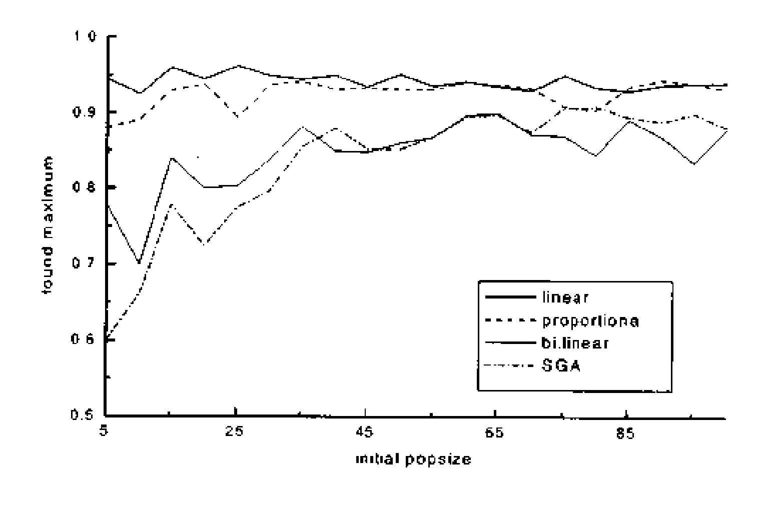


图 4.8 SGA 和 GAVaPS 的比较：初始群体规模对平均执行效果（函数 G4）

对 SGA，低的群体规模值意味着低的消耗和较差的执行效果。一开始增加群体规模可改进执行效果但增加了进行的消耗。有一个“执行效果饱和”阶段，但消耗仍然呈线性增长。对 GAVaPS，初始群体规模对执行效果和消耗都没有实际的影响。

通过分析两个算法对其余的函数 G1~G3 的执行效果和消耗，可得到相似的观察结果。值得注意的是对所有 4 个问题，SGA 对不同的群体规模值存在着最优行为（最好的执行效果，最低的消耗）。而 GAVaPS 采用的是随问题及随搜索状态而变化的群体规模。表 4.1 列出了对所有测试函数 G1~G4 的实算的性能和模拟消耗。对 SGA 和分别以比例、线性、双线性进行寿命分配的 GAVaPS 找到的最好值为 V，函数评价次数为 E。SGA 对函数 G1~G4 的最优群体规模分别是 75, 15, 75, 100。

表 4.1 三种方法的比较

算 法	函数							
	G1		G2		G3		G4	
	V	E	V	E	V	E	V	E
SGA	2.814	1467	0.875	345	1.996	1420	0.959	2186
GAVaPS(1)	2.831	1708	0.875	970	1.999	1682	0.969	2133
GAVaPS(2)	2.841	3040	0.875	1450	1.999	2813	0.970	3739
GAVaPS(3)	2.813	1538	0.875	670	1.999	1555	0.972	2106

线性方案(2)的特点是执行效果最好但消耗也最高。双线性方案(3)消耗最少，但执行效果不如线性好。比例方案(1)则是中等的执行效果，中等的消耗。应该注意到，具有任何(1)~(3)寿命分配方案的 GAVaPS 算法对多数测试情况都比 SGA 提供更好的执行效果。相比 SGA，GAVaPS 的消耗要更高。表中报告的 SGA 结果是在最优群体规模下执行的结果。如对函数 G2，如果对 SGA 实算的群体规模是 75，而不是最优值 15，那么，SGA 的模拟消耗将为 1035。

有关如何选择遗传算法的参数的知识还是很不完整的，有很强的经验性。在这些参数中，群体规模看起来是最重要的，因为它较大程度地影响着遗传算法的模拟消耗。对其设定的最好的方法可能是根据遗传算法的实际需要让它自我调节。这就是 GAVaPS 方法：在搜索过程的不同阶段，不同的群体规模可能是最优的。但所报告的实算结果还很初级，寿命参数的分配仍需更进一步的研究。

4.5 遗传算法、约束及背包问题

正如引言中所论述的，遗传算法的约束处理技术可以分成几类。一种处理违反约束条件对象的方法是不考虑约束条件地去产生潜在解，然后通过降低评价函数的“好坏度”对其进行惩罚。换句话说，约束问题通过引入对违反约束条件的惩罚将问题转换为非约束问题：这些惩罚包含在评价函数里。当然，可使用各种各样的罚函数。一些罚函数分配一个常数作为惩罚尺度。其他罚函数依赖于违反度：违反度越大，惩罚越重。罚函数的生长可以是对数的、线性的、二次方的、指数的，等等。

另外一种惩罚方法是从群体中消去不可行解，即最严重的死亡惩罚。这种技术成功地应用于数值优化问题的演化策略（第 8 章）。但这种方法有其缺点。对一些问题，通

过标准的遗传因子产生可行解的可能性十分小，而使算法耗费大量的时间去评价非法个体。而且，这种方法不分配非可行解到任何群体的基因池(gene-pool)中。

另一类约束处理的方法是用特殊的修补算法来校正所有产生的不可行解。同样，这种修补算法计算上是耗时间的，且最终的算法必须能够适合特定的应用。而且对一些问题，校正解的过程可能和解原始解一样困难。

第三种方法主要是用特殊的表达映射（解码器），它能保证——至少是增加可行解产生的几率，或者是用与问题有关的算子来保存可行解。但是，解码器的运行需要频繁、大量的计算[73]，这种方法不能解决所有的约束，而且导出的算法只能适合特定的应用。

在这一节，我们尝试用上述技术来处理一个特定的0/1背包问题。问题很容易用公式表达，其答案属于NP完全问题中的一种。评价对这类特定单约束问题的约束处理技术的优缺点是一个有趣的练习：结论可以应用于许多约束组合优化问题。注意，本节的主要意图是用一个特殊的例子来说明解码器、修补算法及罚函数的概念(在引言中有简单的讨论)，而不是对所有方法的完整的概述。出于这种理由，我们不提供测试例子的最优解：而只是对现有方法进行一些比较。

4.5.1 0/1 背包问题及测试数据

有很多种背包问题(knapsack problem)，在此类问题中，给出一套实体及它们的价值和尺寸，意图是选择一个或多个互不相干的子集使每个子集的尺寸不超过给定边界，而被选择值的总和最大[252]。这类问题中的许多都是NP难解问题，而且这类问题的许多实例都只能用启发算法来解。用于实算的问题都是0/1背包问题。任务是，对一给定权值(weight)集合 $W[i]$，利益(profits)集合 $P[i]$及容量 C (capacity)，求一个二进制向量 $\bar{x}=\langle x[1],\ldots,x[2]\rangle$，使

$$\sum_{i=1}^{n} x[i]\cdot W[i]\le C$$

且最大化

$$P(x)=\sum_{i=1}^{n} x[i]\cdot P[i]$$

正如前面所显示的，在本节，我们将分析几个以遗传算法为基础的算法及其在随机产生的测试问题中的实算行为。当然这类问题的难度在很大程度上受利益和权值的相关性影响[252]，三套随机产生的数据是：

- 无相关性：

$$W[i]:=(\text{均匀地})\ \mathrm{random}([1..v])$$
$$P[i]:=(\text{均匀地})\ \mathrm{random}([1..v])$$

- 弱相关性：

$$W[i]:=(\text{均匀地})\ \mathrm{random}([1..v])$$
$$P[i]:=W[i]+(\text{均匀地})\ \mathrm{random}([-r..r])$$

（如果对某些 i, $P[i]\le 0$，则利益值被忽略，计算重复进行直到 $P[i]>0$ 为止）。

- 强相关性：

$$W[i] := \text{(均匀地) random}([1..v])$$

$$P[i] := W[i] + r$$

较高的相关性意味着较小的差值：

$$\max_{i=1..n}\{P[i]/W[i]\} - \min_{i=1..n}\{P[i]/W[i]\}$$

正如[252]中所报告的，高相关问题具有较高的预测难度。

数据由下面设定的参数产生：$v=10$ 及 $r=5$。每种类型的测试分别使用了三套数据集合，分别为 n=100、250 及 500 项。根据[252]中的建议，考虑了两种背包的类型：

- 约束容量背包

容量为 $C_1 = 2v$ 的背包。在这种情况下，最优解包含很少几项。不满足条件的域几乎占据整个域。

- 平均容量背包

容量为 $C_2 = 0.5\sum_{i=1}^{n} W[i]$ 的背包。在此情况下，有大约一半的项在最优解里。

如[252]中报道，进一步增加容量值 C 不会较大地增加此经典算法的计算机时。

4.5.2　算法的描述

用三种类型的算法进行了实算测试：基于罚函数的算法 $A_p[i]$（其中 i 是这类特殊算法中的索引值）、基于修补方法的算法 $A_r[i]$和基于解码器的算法 $A_d[i]$。下面依次讨论这三类算法。

算法 $A_p[i]$

在所有这类基于罚函数的算法中，长度为 n 的二进制串表示问题的解 $\mathbf{x}$：如果 $x[i]=1$，第 i 项被选择为背包。每个串的适应值 $eval(\mathbf{x})$是这样确定的：

$$eval(\mathbf{x}) = \sum_{i=1}^{n} x[i]\cdot P[i] - Pen(\mathbf{x})$$

这里罚函数 $Pen(\mathbf{x})$对所有可行解 $\mathbf{x}$ 为零，即 $\sum_{i=1}^{n} x[i]\cdot W[i] \le C$ 的解；否则大于零。

分配罚值的方案有许多。这里只考虑三种情况，它们是：罚函数的增长是违反度的对数，线性及二次方函数：

- $A_p[1]$：$Pen(\mathbf{x}) = \log_2(1+\rho\cdot(\sum_{i=1}^{n} x[i]\cdot W[i] - C)$
- $A_p[2]$：$Pen(\mathbf{x}) = \rho\cdot(\sum_{i=1}^{n} x[i]\cdot W[i] - C)$
- $A_p[3]$：$Pen(\mathbf{x}) = (\rho\cdot(\sum_{i=1}^{n} x[i]\cdot W[i] - C))^2$

对所有三种情况，$\rho = \max_{i=1..n}\{P[i]/W[i]\}$

1. 算法 $A_r[i]$

和前面的算法一样，长度为 n 的二进制串代表问题的解 $\boldsymbol{x}$：当且仅当 $x[i]=1$ 时，第 i 项被选择为背包。每个串的适应值 $eval(\boldsymbol{x})$是这样确定的：

$$eval(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}'[i] \cdot P[i]$$

这里，向量 $\boldsymbol{x}'$ 为原始向量 $\boldsymbol{x}$ 的修补版本。

这里有两点是很有趣的：第一，我们可以考虑不同的修补方法；第二，群体中修补后的染色体可以替换原始染色体的百分数。该替换率可以从 0%到 100%；最近 Orvosh 和 Davis[301]报告了所谓的 5%法则：如果替换原始染色体的概率为 5%，算法的执行要好于其他替换率，特别是好于“不替换”和“全替换”的方案。

我们对这两种不同的修补算法运行和测试。两种算法过程相同，如图 4.9 所示。

```
procedure repair (x)
begin
    knapsack-overfilled := false
  x' := x
  if ( Σ_{i=1}^{n} x'[i]·W[i] > C)
    then knapsack-overfilled := true
        while (knapsack-overfilled) do
        begin
            i := select an item from the knapsack
            remove the selected item from the knapsack: i. e., x'[i]:=0
         if ( Σ_{i=1}^{n} x'[i]·W[i] ≤ C)
            then knapsack-overfilled := false
        end
end
```

图 4.9 修补过程

这里，两种修补算法只是 select 过程不同，它从背包中选一项移走：

- **$A_r[1]$**（随机修补）

  过程 **select** 从背包中选择一个随机成员。

- **$A_r[2]$**（贪婪修补）

背包中的所有项按它们利益和权值的比率的降序排列。然后过程 **select** 总是选择最后一项删除。

2. 算法 $A_d[i]$

对背包问题，一个可能的解码器是基于整数表达①。这里，对选择项我们使用一般的表达（该表达详见第 10 章）。每个染色体都是有 n 个整数的向量：向量的第 i 个组分是一个从 1 到 $n-i+1$ 之间的随机整数。序号表达引用了一个项表 L；一个向量通过选择

① 当然也有其他可能，例如用二进制表达，按下面方式从左到右（即从 i=1 到 n）翻译串：如果(1) $i=1$，且(2)背包中有放该项的地方，则取第 i 项。这种解释总能产生可行解。而且，如果项根据其利益和权值的比率排序，则所有 1 的解对应于贪婪算法的解。这种解码器的例子见第三部分的 15.3 节。

当前项表中适当项被解码。如，对一个项表 $L=(1,2,3,4,5,6)$，向量 $\langle 4,3,4,1,1,1\rangle$ 按照下面的序列解码：4, 3, 6 (因为 6 是当前表中 4 和 3 被移走后的第 4 个成员), 1, 2 和 5。很明显，用这种方法，一个染色体可被解释成将项并入解的方案。另外，应用单点杂交于两个可行的父代将产生可行的后代。变异算子可以用和二进制表达相似的方法定义：如果第 i 个基因被变异，它将采用一个在域$[1, n-i+1]$中均匀分布的随机值。

解码算法如图 4.10 所示。

```
procedure decode (x)
begin
        build a list L of items
        i := 1
        WeightSum := 0
        ProfitSum :=0
        while i ≤ n do
        begin
                j := x[i]
                remove the j item from the list L
                if WeightSum + W[j] ≤ C   then
                begin
                        WeightSum :=WeightSum + Weight[j]
                        ProfitSum :=ProfitSum + Profit[j]
                end
                i :=i +1
        end
end
```

图 4.10　对序号表达的解码过程

这里考虑的基于解码技术的两个算法只是在过程 **build** 上有所不同：

- $A_d[1]$（随机解码）过程 **build** 产生一个项表 L 使表中项的次序对应于输入文件（随机的）中项的次序。
- $A_d[2]$（贪婪解码）过程 **build** 产生一个项表 L 按照它们利益和权值的比率下降的次序。向量 **x** 的解码是在排序的基础上完成的，它同 $A_r[2]$法有些相似。如，$x[i]=23$ 被解释为当前表 L 按照利益和权值比率的降序中的第 23 项。

4.5.3　实算与结果

在所有的实算中，群体规模都等于 100。变异率和杂交率分别固定为 0.05 和 0.65，我们收集了 500 代里的最好解作为对算法执行结果的度量。一些经验已经证实经过这样的代数后，已观察不到改进。表 4.2 中报告的结果是 25 个实算的平均值。精确解没有列入表中；此表只是比较了不同算法的相对效率。注意，数据文件没有排序，即为任意的项序，与它们的 $P[i]$ / W[i]比率无关。容量型 C_1 和 C_2 分别为代表约束容量和平均容量（第 2 节）。

用 5%修补法则获得了方法 $A_r[1]$和 $A_r[2]$的结果。我们同样检查了是否 5%法则能很好地解 0/1 背包问题（该法则是在两个组合问题的实算时发现的：网络设计问题和图像着色问题[301]）。出于比较的意图，我们选择了权值和利益弱相关性的测试数据集合。所有设定参数都是固定的，修补率的值从 0% 到 100%。可以观察到 5% 法则对遗传算法的执行没有影响。算法 $A_r[2]$的结果列在表 4.3 中。

表 4.2 实算结果：符号“*”表示在给定时间限制内没有发现有效解

相关性	项数	容量型	方法						
			$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$	$A_d[1]$	$A_d[2]$
无	100	C_1	*	*	*	62.9	94.0	63.5	59.4
		C_2	398.1	341.3	342.6	344.6	371.3	354.7	353.3
	250	C_1	*	*	*	62.6	135.1	58.0	60.4
		C_2	919.6	837.3	825.5	842.2	894.4	867.4	857.5
	500	C_1	*	*	*	63.9	156.2	61.0	61.4
		C_2	1712.2	1570.8	1565.1	1577.4	1663.2	1602.8	1597.0
弱	100	C_1	*	*	*	39.7	51.0	38.2	38.4
		C_2	408.5	327.0	328.3	330.1	358.2	333.6	332.3
	250	C_1	*	*	*	43.7	74.0	42.7	44.7
		C_2	920.8	791.3	788.5	798.4	852.1	804.4	799.0
	500	C_1	*	*	*	44.5	93.8	43.2	44.5
		C_2	1729.0	1531.8	1532.0	1538.6	1624.8	1548.4	1547.1
强	100	C_1	*	*	*	61.6	90.0	59.5	59.5
		C_2	741.7	564.5	564.4	566.5	577.0	576.2	576.2
	250	C_1	*	*	*	65.5	117.0	65.5	64.0
		C_2	1631.9	1339.5	1343.4	1345.8	1364.4	1366.4	1359.0
	500	C_1	*	*	*	67.5	120.0	67.1	64.1
		C_2	3051.6	2703.8	2700.8	2709.5	2748.1	2738.0	2744.0

表 4.3 修补率对算法 $A_r[2]$ 执行效果的影响

相关性	弱					
项 数	100		250		500	
容量型	C_1	C_2	C_1	C_2	C_1	C_2
修补率						
0	94.0	371.3	134.1	895.0	158.0	1649.1
5	94.0	371.7	135.1	891.4	155.8	1648.5
10	94.0	370.2	135.1	889.5	157.3	1640.3
15	94.0	368.3	135.3	895.4	156.6	1646.2
20	94.0	372.0	135.6	905.7	155.8	1643.6
25	94.0	370.2	135.1	894.0	155.8	1644.0
30	94.0	367.2	135.1	895.7	157.3	1648.0
35	94.0	370.3	136.1	896.5	156.3	1643.3
40	94.0	368.6	134.3	886.5	156.1	1648.4
45	94.0	369.0	135.3	891.5	156.6	1649.0
50	94.0	371.7	134.6	891.0	156.1	1641.5
55	94.0	371.3	135.0	895.7	157.0	1647.0
60	94.0	369.6	135.0	894.0	156.1	1645.0
65	94.0	370.0	135.1	893.2	156.6	1642.8
70	94.0	367.6	135.0	893.4	156.1	1640.9
75	94.0	367.7	135.3	895.7	157.1	1648.1
80	94.0	368.2	134.3	898.5	155.8	1648.5
85	94.0	364.7	135.6	897.4	156.1	1646.2
90	94.0	368.7	134.3	885.2	156.1	1648.9
95	94.0	371.2	135.0	890.5	155.3	1642.3
100	94.0	370.2	134.6	901.0	156.3	1646.1

实算得到的主要结论总结如下：

- 罚函数 $A_p[i]$ 对所有 i 并不能获得约束背包容量(C_1)问题的可行解。这一结果对任何数目项（n =100，250 和 500）和任何相关性的情况都一样。
- 仅从对有平均背包容量(C_2)问题的实算结果来看，基于对数罚函数的算法 $A_p[1]$具有明显优势：在所有情况下（不相关、 弱相关和强相关， 项数 n = 100，250 和 500），其执行结果都压倒其他技术。但是，如前所述，对约束容量问题是不成功的。
- 仅从约束背包容量(C_1)问题的实算结果判断，修补方法 $A_r[2]$（贪婪修补）在所有的测试情况下比其他方法好。

这些结论都是很直觉性的，在约束背包容量的情况下，只有很小一部分可能项的子集组成可行解；因此，多数罚函数都失败。对许多其他组合问题都存在这种情况：搜索空间可行部分与整个搜索空间的比率越小，罚函数的方法越难提供可行解。这在[332]中已有评述，其作者认为：

> “对稀疏问题，粗糙的罚函数很难找到解。即使偶然找到，解也很差。理由很明显：对一个稀疏可行区域，初始群体不可能包含任何解。因为粗糙的罚函数对不可行解是不加分辨的，遗传算法会毫无目的地徘徊。如果提供一个幸运的变异或者奇特的杂交，一个后代碰巧落在可行区域，该子代将变成一个超级个体，其遗传材质将很快占据整个群体并确定无疑地过早收敛。”

从另一方面来看，修补算法又确实执行得很好。在平均背包容量的情况下，对数罚函数法（即小惩罚）要更优越一些：可以注意到，问题的大小不影响结论。

如前所述，这些实算不提供全貌：许多其他实算将在不久的将来计划进行。在罚函数的类型里，实算其他附加规则可能是有趣的。所有考虑的惩罚都是 $Pen(\mathbf{x}) = f(\mathbf{x})$的形式，这里 f 为对数、线性和二次方。其他方法可能用 $Pen(\mathbf{x}) = a + f(\mathbf{x})$实算（$a$ 为一些常数）。这将对所有可行向量进行最小惩罚。更进一步，实算动态罚函数可能是很有趣的，这里它们的值依赖于附加参数，像代数（这项工作在[267]中用在对连续变量的数值优化）或者是搜索的特征性[360]，其中的动态罚函数被用在设备的布局问题：当更好的可行解或非可行解找到时，强加在非可行解上的惩罚将变化）。这看起来是一项有价值的自适应罚函数实算。总之，应用算子的概率可能是适应的（和演化策略一样）；一些初步实算表明适应群体规模可能有一些优点（4.4 节）；所以适应罚函数的想法值得注意。在其最简单的版本里，一个惩罚系数成为解向量的部分，并且经历所有的遗传（随机）变换，和动态罚函数的思想相对，这些罚系数规则地变换，例如为一代数的函数。

也可以实算许多其他修补方案，包括其他启发法，而不是利益和权值的比率。把惩罚法和修补算法结合起来也是可能的：算法 $A_p[i]$的非可行解可能被修补成可行解。

对解码器一类算法，有必要实算不同的整数表达（如第 10 章的货郎担问题）：邻接表达（带有选边杂交、子巡回杂交或者是启发杂交），或者是路径表达（带有 PMX，OX 和 CX 杂交，或边缘重组杂交）。比较这些表达和算子对 0/1 背包问题的适用性是很有趣的，如货郎担问题和调度问题[370]。一些新的与问题有关的杂交将提供很好的结果。

4.6 其他思想

在本章的前面部分，我们已经讨论了一些与消除样本机制的可能错误有关的论点。这些研究的基本目的是增强遗传搜索；特别是阻止遗传算法过早收敛。近些年，已有不同的方法来加强模式处理过程，本节将讨论其中的一些方法。

第一个方向涉及到遗传杂交算子。该算子是受生物过程的启发，但是也存在一些缺点，例如，假定有两个较好的模式：

$$S_1 = (0\,0\,1 * * * * * * * * * 0\,1) \text{ 和 } S_2 = (* * * * 1\,1 * * * * * * * *)$$

在一群体中同样有两个向量串，v_1 和 v_2，分别被 S_1 和 S_2 匹配：

$$v_1 = (\mathbf{0\,0\,1}\,0\,0\,0\,1\,1\,0\,1\,0\,\mathbf{0\,1}) \text{ 和 } v_2 = (1\,1\,1\,0\,\mathbf{1\,1}\,0\,0\,0\,1\,0\,0\,0)$$

很明显，杂交不能对染色体中解码后的某些特征进行组合，也不可能获得被模式

$$S_3 = (0\,0\,1 * 1\,1 * * * * * 0\,1)$$

所匹配的串，因为第一个模式被毁坏。

另一个对经典的单点杂交提出质疑的观点是变异和杂交的不均匀性：变异依赖于染色体的长度，而杂交不是。例如，假设变异率为 $p_m = 0.01$，染色体的长度为 100，预计染色体上的变异位数是一个。如果染色体的长度为 1000，预计染色体上的变异位数则为 10。而在两种情况下，单点杂交都是在杂交位置的基础上组合两个串，而不管串的长度。

一些研究者[104,382]实算了其他不同的杂交方法，例如，两点杂交选择两个杂交位置对两点之间的染色体材质进行交换。很明显，串 v_1 和 v_2 可能产生一对后代：

$$v_1' = (\mathbf{0\,0\,1}|0\,\mathbf{1\,1}\,1\,0\,0|0\,1\,0\,\mathbf{0\,1}) \text{和 } v_2' = (1\,1\,1|0\,0\,0\,1\,1|0\,1\,0\,0\,0),$$

这里 v_1' 被

$$S_3 = (0\,0\,1 * 1\,1 * * * * * 0\,1)$$

匹配，这对单点杂交是不可能得到的。

同样，有一些模式是两点杂交所不能获得的。我们自然可以实算多点杂交[104]。注意，因为多点杂交必须在两个亲体中互换片断（将一个染色体切割成 s 片后获得），片断必须是均匀的，即多点杂交不是单点杂交的一个自然而然的扩展。

Schaffer 和 Morishima[341]用同样的过程，即适者生存和重组，实算了一个采用杂交点分配的杂交。这是通过向串表达中引入特殊标记完成的。这些标记跟踪串杂交的位置。希望的是杂交位置也经历一个交换过程：如果特定位置产生不好的后代，该位就死亡，否则就生存。实算表明[341]适应杂交对一套测试问题比经典的遗传算法执行结果要好。Spears[367]通过附加位扩展染色体表达实算了采用特定杂交的结果，其中考虑了两种杂交：单点杂交和均匀杂交。

一些研究者[104]实算了片断杂交和混合杂交等其他杂交。片断杂交是一个多点杂交的变种，它允许杂交点数变化，固定数目的杂交点或片断被一个片断交换率所代替。该交换率指明一个片断将在串中的任意点结束的概率。例如，如果片断交换率为 $s = 0.2$，那么，从片断的开始点起，该片断在每个点上终止的机会为 0.2。换句话说，如果片断交换

率 $s=0.2$，片断的预计数目将是 $m/5$，不像多点杂交，杂交点数目是变化的。混合杂交可以被看成是应用其他杂交的附加机制。混合杂交：(1)随机地混合前后排列的两个串的位位置；(2)杂交串，即在杂交点之间交换片断；(3)不混合串。

对单点杂交，两点杂交和多点杂交的更进一步的总结是均匀杂交(uniform crossover)[366, 382]。第一个后代的每一位用某一概率 p 决定哪一个亲体将在该位贡献值。第二个后代将从另一个亲体中的该位中接受值。

例如，$p=0.5$ (0.5 –均匀杂交)，串

$$v_1=(0\,0\,1\,0\,0\,0\,1\,1\,0\,1\,0\,0\,1) \text{ 和 } v_2=(1\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,0\,0)$$

可能产生下面的后代：

$$v_1'=(0_1\,0_1\,1_2\,0_1\,1_2\,1_2\,0_2\,1_1\,0_1\,1_2\,0_1\,0_1\,0_2) \text{ 和}$$

$$v_2'=(1_2\,1_2\,1_1\,0_2\,0_1\,0_1\,1_1\,0_2\,0_2\,1_1\,0_2\,0_2\,1_1)$$

这里，下标 1 和 2 分别代表对应于向量 v_1 和 v_2 亲体号。如果 $p=0.1$ (0.1–均匀杂交)，两个串 v_1 和 v_2 可能产生

$$v_1'=(0_1\,0_1\,1_1\,0_1\,1_2\,0_1\,1_1\,1_1\,0_1\,1_2\,0_1\,0_1\,1_1) \text{ 和}$$

$$v_2'=(1_2\,1_2\,1_2\,0_2\,0_1\,1_2\,0_2\,0_2\,0_2\,1_1\,0_2\,0_2\,0_2)$$

因为均匀杂交是交换位而不是片断，所以，它可以不考虑相对位置而交换特征。对一些问题[382]，这种能力压倒了破坏基因块所带来的缺点。但是，对其他问题，均匀杂交要比两点杂交差。Syswerda[382]从理论上比较了 0.5–均匀杂交、单点杂交和两点杂交 Spears 和 De Jong[366]给出了 p–均匀杂交的分析，即平均包含 $m \cdot p$ 个杂交点的杂交。

Eshelman [104]报告了几个杂交算子的实算。结论表明单点杂交最差：但也没有明显的赢家。对上述实算的总结是：这些杂交中的每一个对某一类特定问题有用，对其他问题却效果不好。正是这些依赖于算子的问题触发了演化程序的思想。

Muhlenbein 和 Voigt[290]研究了称为基因池重组(gene pool recombination)的新重组算子的特性，其中基因是随机地从被选亲体所定义的基因池中挑出的。该算子的一个有趣之处是所谓的过量性：几个亲体产生一个后代。Eiben[100]同样研究了这种多亲杂交(multi-parent crossover)，其中考虑了几种基因扫描技术以从几位亲体中产生一个后代。Renders 和 Bersini[324]实算了单纯杂交(simplex crossover)的数值优化问题：该杂交包括计算亲体群的质心(centroid)，并从远离质心点的最差个体移开。分散搜索(scatter search)技术[142]也建议使用多亲体。

一些研究者研究了遗传算法的控制参数（如群体规模、算子概率）对系统执行的效果。Grefenstette[159]使用了一个 meta-GA 去控制另一个 GA 的参数。Goldberg[153]给出了最优群体规模的理论分析。一个控制参数对遗传搜索的影响的完整研究（函数优化的在线性能）在[343]中有论述。研究表明：（1）变异起着比以前公认的背景算子作用更强的作用；（2）杂交率的重要性比预计的要小；（3）基于选择和变异的搜索方案可能是强大的搜索过程，尽管没有杂交的帮助（如第 8 章出现的演化策略）。但是，遗传算法仍然缺乏好的启发方法来确定其参数值：仍然没有一个对所有考虑的问题都适合的环境。看起来寻找遗传算法参数的最好值仍然更像一门艺术，而不是一门科学。

到本章为止，我们已经讨论了两个基本的遗传算子：杂交（单点、两点、均匀等）和变异，它们都是以固定的比率（杂交率 p_c 和变异率 p_m）应用于个体或单个位上的。正如从第 2 章的运行例子中所看到的，将杂交和变异用于同样个体是可能的（如 v_{13}）。实际上，这两个算子可以被看成一个组合算子，“杂交和变异”算子，因为这两个算子都可以同时用于个体。实算遗传算子的一种可能是使它们独立：在重组阶段应用这两个算子中的一个或者另一个，但不是全部[78]，这种分离有一些优点。首先，变异不再被用于杂交算子处理后的个体，使整个过程在概念上比较简单；第二，很容易加入新的算子；如几个与问题有关的算子。这就是演化程序的思想：有许多被用于个体数据编码的与问题有关的“遗传算子”。回忆一下，对本书中的演化程序，我们开发了一个新的特殊的选择规则 modGA（前一节），它促成了上述思想。而且，我们可以更深入一步。每一个算子都有其自己的适应值，它将同样经历一些演化过程。算子的选择和应用根据其适应值是随机的，这些思想并不是最新的，它们以前就已出现[77,115,78]。但是，它在演化规划技术中被赋予了新的含义和重要性。

另一个有趣的方向是搜索更好的模式处理过程，这在前面的章节中和欺骗问题一起进行了论述，最近的建议是：散乱遗传算法(mGA, messy Genetic Algorithm)[155,159]。mGA 在许多方面和经典的遗传算法不同：表达、算子、群体规模、选择及评价过程。我们将简要地依次对它们进行讨论。

首先，染色体上的每一位都被其名字所标记——和前面章节中讨论的转置算子技巧相同。另外，串是变长的，不要求串有完整的基因互补成分。一个串可能有多余的甚至是矛盾的基因。例如，下面的串在 mGA 中是正当的：

$v_1 = ((7,1)(1,0))$

$v_2 = ((3,1)(9,0)(3,1)(3,1),(3,1))$

$v_3 = ((2,1)(2,0)(4,1)(5,0)(6,0)(7,1)(8,1))$

每对括弧中的第一个数字表示位位置，第二个数字为位的值，这样，第一个串 v_1 指定了两位：第 7 位置上的位 1 和第 1 位置上的位 0。

为评价这样的串，我们不得不处理过分详细的问题（串 v_3，其中第 2 位置上被指定了两个位）和太不详细的问题（所有的三个向量都不确定的情况，假定为第 9 位置）。对于过分详细的处理可以用许多方法：例如，可以使用一些确定的或者随机的投票过程。而太不详细的问题是较难处理的，有兴趣的读者可以参阅[155]、[158]、[159]。

很明显，变长度、过分详细的和太不详细的串将影响所使用的算子。简单的杂交可以被两个甚至是更简单的算子所替换：接合(splice)和切割(cut)。接合算子以特定的接合概率连接两个被选择串。例如，用 v_2 接合串 v_1 我们得到：

$v_4 = ((7,1)(1,0)(3,1)(9,0)(3,1)(3,1),(3,1))$

切割算子以某一切割概率在随机确定的位置切割被选择串。如在位置 4 切割串 v_3，可以得到：

$v_5 = ((2,1)(2,0)(4,1)(5,0))$ 和 $v_6 = ((6,0)(7,1)(8,1))$

另外，有一个不变的变异算子，即以指定概率从 0 变 1，或从 1 变 0。GA 和 mGA

有些不同，散乱遗传算法（对可靠的选择而不管函数的尺度）使用了竞争选择的方式[155]；它们同样把演化过程划分成两个阶段：第一阶段选择基因块，只在第二阶段使用遗传算子，而且群体规模在过程中是变化的。

散乱遗传算法对几个欺骗函数的测试得到很好的结果[155,158]。正如 Goldberg[155]所表述的：

> “对一个被设计得很难的测试函数，……在两套实算的运行中，mGA 收敛到测试函数全局最优解，比较而言，一个使用随机次序串的简单的 GA 只能得到 25% 的正确解。”

[158]中论述道：

> “因为 mGAs 能使最差条件下的问题收敛，相信它们将发现所有其他有界欺骗问题的全局最优解，而且，mGAs 在计算时间上被构造成收敛的，计算机时的增加对串行机只是决定变量数目的多项式函数；对并行机，只是决定变量数目的对数函数。最后，mGAs 是一个实用的工具，用以登上欺骗函数阶梯，并沿着这条路为我们提供有用的而且相对便宜的中间结果。”

还有其他一些试图增强遗传搜索的方法。一个修正的遗传算法，称之为 Delta 解码最近由 Whitley[400]等建议。Schraudoph 和 Belew[347]建议了一个动态测试解码(DPE)方法，其中解码个体的精度是动态调节的。这些算法将在本书的第 8 章进行讨论。



第二部分　数 值 优 化



第5章　二进制编码和浮点编码

正如在前面章节中所讨论的，遗传算法的应用会遇到一些问题。如果不设法避免，遗传算法有时会耽搁找到所需精度的最优解。这就意味着整个群体过早收敛到一个非全局最优上（第 4 章）；其他后果包括在执行局部微调上以及当存在非常规约束时操作上的无能（第6，7章）。

遗传算法传统上使用的二进制编码当用于多维、高精度数值问题时，会有一些障碍。例如，对有 100 个变量、域区间[-500, 500]、精度要求精确到小数点后第 6 位的问题，二进制解向量的长度是 3000。这本身会产生一个大约是 10^{1000} 的搜索空间。对这样的问题，遗传算法确实执行得很不好。

二进制字母表提供了任一码位每位信息最大的模式数[154]，因此解的位串表达支配着遗传算法的研究。这种编码的研究也有助于理论分析，并可能产生优美的遗传算子。但是，“隐含并行性”的结果不依赖于所使用的位串[9]，而且值得运算人的字母表和可能的新遗传算子。特别是对变量作用于连续域的参数优化问题，我们可用实型码基因连同相应的特殊“遗传”算子运算 。

在参考文献[157]中，Goldberg 写道：

“在人工遗传和演化搜索方案中，实型编码或浮点基因的使用已经有很长的历史了，尽管是有争议的，但它们在以后的使用趋势仍在上升。这种上升的使用趋势多少有些使熟知基本遗传算法（GA）理论的研究者感到惊讶[154,188]，因为简单的分析似乎建议通过使用低基数性的字母表可以增强模式的处理，而来自于实算的结论似乎直接反驳了这种观点，实型编码在许多实际问题里工作得较好。”

本章描述使用浮点编码及修正遗传算子的实算结果。这样做的主要目的是使遗传算法更移近问题空间（演化规划的思想）。通过利用真实空间的一些特征，这样的移近强制（但不总是）算子更与问题的特殊性有关。例如，编码有这样的性质：在表达空间相互靠近的两个点也必须在问题空间里靠近，反之亦然。对二进制方法却并不总是这样，其中表达的距离一般用不同位上的数定义。但是用灰码（Gray coding）可能能减少这样的差异。

表 5.1　二进制正码和灰码

二进制正码	灰码	二进制正码	灰码
0000	0000	1000	1100
0001	0001	1001	1101
0010	0011	1010	1111
0011	0010	1011	1110
0100	0110	1100	1010
0101	0111	1101	1011
0110	0101	1110	1001
0111	0100	1111	1000

```
procedure Binary-to-Gray
begin
    g_1 = b_1
    for  k=2 to m do
        g_k = b_{k-1} XOR b_k
end

procedure Gray-to-Binary
begin
    value = g_1
    b_1 = value
    for k=2 to m do
    begin
        if g_k = 1 then value = NOT value
        b_k = value
    end
end
```

图 5.1 二进制正码－灰码和灰码－正码的转换过程

转换一个二进制正码数 $\boldsymbol{b}=\langle b_1,\ldots,b_m\rangle$ 到灰码数 $\boldsymbol{g}=\langle g_1,\ldots,g_m\rangle$及相反的过程如图 5.1 所示；参数 m 表示这些编码中的位数。表 5.1 列出了前 16 个二进制正码数和相应的灰码。

注意，灰码编码有一个性质：任何两个在问题空间彼此相邻的点只有一位不同。换句话说，参数值增加一步相应于代码改变一单个位。同时要注意也存在其他类似的转换二进制正码和灰码的方法。例如，（$m=4$ 的情况），一对矩阵：

$$A=\begin{bmatrix}1&0&0&0\\1&1&0&0\\0&1&1&0\\0&0&1&1\end{bmatrix}\qquad A^{-1}=\begin{bmatrix}1&0&0&0\\1&1&0&0\\1&1&1&0\\1&1&1&1\end{bmatrix}$$

给出了下面的转换：

$$\boldsymbol{g}=A\boldsymbol{b} \text{ 和 } \boldsymbol{b}=A^{-1}\boldsymbol{g}$$

其中乘法运算是以 2 为模完成的。

我们还是使用浮点表达，因为它概念上更靠近问题空间，同时也能容易而有效地允许封闭的、动态的算子的执行（见第 6 章，第 7 章）。随后，我们用各种新算子对许多测试例子比较了二进制和浮点的执行情况。本章将涉及到说明对一个典型的动态控制问题测试时，二进制和浮点表达之间的不同。这是一个线性约束二次问题，它是我们下一章要讲到的一个问题的特殊例子（还有两个其他动态控制问题），用以说明演化程序的进展、过早收敛及局部微调。正如所预料的，结果要好于二进制表达。其他研究者也获得同样的结论[78,408]。

5.1 测试例子

在实算中，我们选择了下面的动态控制问题：

$$\min\left(x_N^2+\sum_{k=0}^{N-1}(x_k^2+u_k^2)\right)$$

并服从　　$x_{k+1}=x_k+u_k$，$k=0,1,\ldots,N-1$

其中 x_0 是给定的初始态，$x_k\in R$ 是一个状态，$u\in R^N$ 为寻找的控制向量。最优值可以被解析成：

$$J^*=K_0x_0^{\ 2}$$

其中，K_k 是 Riccati 方程的解：

$$K_k=1+K_{k+1}/(1+K_{k+1})\ 且\ K_N=1$$

在实算中，一个染色体表达控制状态 u 的一个向量。同时假定对每个 u_i，都有一个固定的域 $\langle -200,200\rangle$（对所执行的测试类，实际的解落在此区间里）。本章以后所有的实算都使用 $x_0=100$ 及 $N=45$，即染色体 $u=\langle u_0,\ldots,u_{44}\rangle$ 有最优值 $J^*=16180.4$。

5.2 两 种 执 行

对此研究我们选择了两种遗传算法的执行，它们所不同的只是表达和使用的遗传算子，其他都相同。这样的方法给我们一个更直接的比较。两种执行都使用了相同的选择机制：随机全域取样[23]。

5.2.1　二进制执行

在二进制执行中，一个染色体向量中的每个元素都使用相同的位数编码。为利于更快地运行时解码，每个元素只占据其自身的记忆词的容量（通常，如果每个元素的位数超过词大小，它将占据不止一位，但是这种情况很容易扩展）：用这种方法，元素可以被当作整数存取，这就消除了二进制到十进制的解码需要。然后每个染色体都是 N 个词的一个向量，它等于每个染色体中的元素数，当要求用多重词表达所需位数时，则是倍数关系。

这种方法的精度对一个有限固定域依赖于实际使用的位数，且等于 $(UB-LB)/(2^n-1)$，其中 UB 和 LB 是上界和下界，n 是染色体中每个元素的位数。

5.2.2　浮点执行

在浮点（FP）执行中，每个染色体向量被编码成一个浮点数向量，解向量有相同的长度。每个元素强制在要求的范围里，而且算子被精心设计以遵循这种要求。

这种方法的精度依赖于使用的机器，但它总体上比二进制表达好得多。当然，我们总是通过引入更多的位扩展二进制表达的精度，但是这将使算法速度显著放慢（参见 5.4 节）。

另外，FP 能够表达十分大的域（或者未知域）。反之对给定二进制长度的二进制表达，增加域范围要以牺牲精度为代价。同时，FP 表达更容易设计处理非常规约束的特殊工具：这将在第 7 章中详细讨论。

5.3 实 算

实算是在 DEC3100 工作站上完成的。这里给出的所有实算结果都是 10 次独立运行的平均结果。在所有的实算中，群体规模保持为 60，迭代次数设定为 20000。除非特别提到，二进制表达使用 $n = 30$ 位来编码一个变量（解向量的一个元素），即对整个向量有 30×45 = 1350 位。

因为可能存在对变异算子解释上的差异，我们把染色体的更新概率作为一个比较浮点和二进制表达公正的度量。所有的实算值都是在运行相同比率的算子基础上获得的：因此，一些迭代数目可以交替地近似用相同数量函数评价来处理。

5.3.1 随机变异和杂交

在这一部分，我们用和传统遗传算法相同的算子进行了实算（至少是对二进制编码）。

1. 二进制

二进制的执行使用了传统的变异和杂交算子。为使它们和浮点执行所用的算子更相似，我们允许杂交只在元素之间进行。杂交概率固定为 0.25，而变异概率随实现的染色体更新率而变化，如表 5.2 所示。

表 5.2 染色体更新率对变异率

执行	染色体更新率				
	0.6	0.7	0.8	0.9	0.95
二进制，p_m	0.00047	0.00068	0.00098	0.0015	0.0021
浮点，p_m	0.014	0.02	0.03	0.045	0.061

2. 浮点

杂交算子执行十分类似于二进制，即在浮点数之间划分点，而且使用相同的概率 0.25。变异随机地应用于浮点数字上而不是位上；这样变异的结果是域〈LB, UB〉里的一个随机值。

表 5.3 各种染色体更新率下的平均结果

执行	染色体更新率					标准偏差
	0.6	0.7	0.8	0.9	0.95	
二进制	42179	46102	29290	52769	30573	31212
浮点	46594	41806	47454	69624	82371	11275

3. 结果

见表 5.3，浮点稍微好于二进制；但是远离最优解（16180.4）时很难判断哪个更好。此外出现一个有趣的现象是浮点执行更稳定，标准偏差更低。

另外，上述实算对浮点表达不十分公正；它的随机变异行为比二进制执行的变异更“随机”，在二进制执行中，改变一个随机位并不意味着产生域里一个完全随机的值。作为一个说明，考虑下面的问题：经过变异后，一个元素从其旧值落进范围（400，因为域为⟨-200, 200⟩）里的概率δ%是多少？ 答案是：

[浮　点] 该概率很明显在区间$\langle\delta, 2\delta\rangle$里。例如，对$\delta = 0.05$，它在$\langle 0.05, 0.1\rangle$范围内。

[二进制] 这里我们需要考虑可以安全地改变的低阶位的数目。假定元素的长度 n =30，m 是允许变化的长度，m 必须满足 $m \le n + \log_2\delta$。因为 m 是一个整数，所以，$m = \lfloor n + \log_2\delta \rfloor = 25$ ，得到的概率为 $m / n = 25/30 = 0.833$，与浮点相比，它是一个非常不同的数。

因此在下面部分，我们尝试设计一个方法来补偿这种缺点。

5.3.2 非均匀变异

在实算的这一部分，除了第 5.3.1 节所讨论的算子，我们还运行一个特殊的动态变异算子来改进单元素调节和减少浮点执行中随机变异的缺点。我们称之为非均匀变异；对此算子的完整讨论可以参考下面内容。

1. 浮点

新算子定义如下：如果$s_v^t = \langle v_1,\ldots,v_m\rangle$ 是一个染色体（t 是代数），元素 v_k 被选择变异，结果是一个向量$s_v^{t+1} = \langle v_1,\ldots,v_k',\ldots,v_m\rangle$，其中

$$v_k' = \begin{cases} v_k + \Delta(t, UB - v_k) & \text{若随机数字为 0} \\ v_k - \Delta(t, v_k - LB) & \text{若随机数字为 1} \end{cases}$$

UB 和 LB 为变量 v_k 的上下限。函数 $\Delta(t, y)$返回区间$[0, y]$里的一个值，以便当 t 增加时，$\Delta(t, y)$以接近于 0 的概率增加。这种性质引起此算子初始均匀地搜索空间（当 t 较小时），而在后面阶段则非常局部化：这样就使产生的新数靠近其继承者的概率增加，而不再是随机选择。我们使用下面的函数：

$$\Delta(t, y) = y \cdot \left(1 - r^{\left(1-\frac{t}{T}\right)^b}\right)$$

其中 r 是一个在区间 [0,1]里的随机数，T 是最大代数，b 是确定对迭代数依赖程度的系统参数，这里 b=5。

2. 二进制

为了使二进制编码的实现更公正，我们模拟了动态算子到其空间里，尽管它的引入主要是改进浮点变异的。和浮点类似，但这里使用了定义不同的 v_k'：

$$v_k' = mutate\,(v_k, \nabla(t, n))$$

其中 n =30 是染色体中每个元素的位数；$mutate\,(v_k, pos)$意思是：第 k 个元素在 pos 位上变异其值（0 位是最低有效位），

$$\nabla(t,n) = \begin{cases} \lfloor \Delta(t,n) \rfloor & \text{如果随机数字为 0} \\ \lceil \Delta(t,n) \rceil & \text{如果随机数字为 1} \end{cases}$$

如果想得到相似的行为，Δ的参数 b 需要进行适当的调节，这里 $b=1.5$。

3. 结果

我们用非均匀变异按前面定义相同的比率的变异重复 5.3.1 节相似的实算。

表 5.4 不同染色体更新率下的平均结果

执　行	染色体更新率		标准偏差
	0.8	0.9	
二进制	35265	30373	40256
浮　点	20561	26164	2133

现在，浮点执行显示了一个更好的平均结果（表 5.4）。二进制的结果仍然是相对更不稳定。但是，值得注意的是尽管浮点执行有较高的平均值，二进制执行在这一轮中产生了两个单个的最好结果（16205 和 16189）。

5.3.3 其他算子

在实算的这一部分，我们决定执行并使用在两个表达空间都容易使用的尽可能多的其他算子。

1. 二进制

除了前面已描述的，我们还实算了多点杂交，而且允许元素位间进行杂交。多点杂交算子应用于单个元素的概率由系统参数控制，设定为 0.3。

2. 浮点

这里，我们同样执行了一个相似的多点杂交。另外还实算了单点和多点算术杂交。它们平均了两个元素的值而不是改变元素的值。算子具有这样的性质：新染色体的每个元素仍然在原始域里。有关这些算子更多的细节在下面两章中讨论。

3. 结果

这里，浮点执行显示了杰出的优越性（表 5.5）；虽然平均结果有点差异，浮点执行在完成它们时却是一致的。

表 5.5 平均结果对染色体更新概率

执　行	染色体更新率			标准偏差	最好结果
	0.7	0.8	0.9		
二进制	23814	19234	27456	6078	16188.2
浮　点	16248	16798	16198	54	16182.1

5.4 执行时间

对处理非常规问题的遗传算法，有许多对时间复杂性的抱怨。在这一节，我们将比较两个执行的消耗时间。列在表 5.6 中的是 5.3.3 节问题运行所得的结果。

表 5.6 比较了染色体中各种数目的元素在两种执行下的 CPU 时间。与二进制相比，浮点版本要快得多，即使是与中等的每个变量有 30 个位的二进制执行相比。对大的域和高精度，染色体的总长会增加，相对差异将进一步扩大，如表 5.7 所示。

表 5.6　CPU 时间（秒）对元素的数目

执　行	元素数（N）				
	5	15	25	35	45
二进制	1080	3123	5137	7177	79221
浮　点	184	398	611	823	1072

表 5.7　CPU 时间（秒）对每个元素的位数：N = 45

执　行	每个二进制元素的位数					
	5	10	20	30	40	50
二进制	4426	5355	7438	9219	10981	12734
浮　点	1072（常数）					

5.5 结　论

示范性的实算表明浮点表达更快，在运行之间更一致，并能提供较高的精度，特别是对大的域二进制编码需要长度可怕的表达。同时，其性能还可以通过特殊的算子达到高准确度的增强（甚至高于二进制表达）。另外，浮点表达直觉上更靠近问题空间，更容易设计并入与问题有关知识的其他算子。这对处理非常规、与问题有关的约束是非常必要的，见第 7 章。

结论是和更愿意使用浮点编码的遗传-演化技术的用户的理由[157]是一致的：（1）使用一个基因一个变量的对应很顺手；（2）避免 Hamming 悬崖（Hamming cliffs）和其他人为地将位串处理成无符号二进制整数的变异运算；（3）对群体适应使用较少的代。

读完本章后，我们鼓励读者运行几个实例，如附录 D 的练习 3；选择几个测试函数，可取附录 B 的一些函数，并用三个基于二进制正码、灰码和浮点系统的遗传算法进行实算。对前两个系统，可使用 GENESIS 1.2；对第三个实算可使用已用过的 GENOCOP（第 7 章）。

第6章 局部微调

遗传算法在数值应用执行局部搜索中显露出固有的困难。Holland[188]建议：在一个能利用域的知识指导局部搜索的系统进行搜索前，应该用遗传算法进行初始搜索的预处理。正如[170]中所观察到的：

> "就像自然的遗传系统一样，遗传算法通过变化高性能的子结构在整个群体中的贡献能力来获得进展；而个体结构并不是注意的焦点。一旦搜索空间的高性能区间被遗传算法识别，使用局部搜索路线来优化最后群体成员可能是有益处的。"

局部搜索利用比模式定律所建议的级更高、定义长度更长的模式。对参数域为无穷、参数的数目十分大及精度要求高的问题，意味着二进制解向量的长度是很长的（对 100 个变量、区间[-500, 500]、精度为小数点后六位的问题，二进制解向量的长度为 3000）。正如前面章节所描述的，对这样的问题，遗传算法的执行效果很差。

为改进遗传算法的局部微调能力，这对高精度问题是必不可少的，我们设计了一个特殊的变异算子，其执行效果不同于传统的变异。回忆一下，传统的变异在某一时间改变一个染色体上的一位；因此这样的变换只使用了局部知识——只是经历变异的位。这样的位，如果处于变量编码序列的左边部分，变异将对变量值的绝对数量级产生较大的影响。反之，如果这些位处于序列的右边，则变异只产生较小的影响。我们决定按照下面的方法使用这些与位置有关的全局知识：当群体不断成长时，位于各个变量编码序列较右的位获得较高的变异概率，而位于较左的位使用较小的概率。换句话说，在迭代过程的开始，这样的变异引起搜索空间的全局搜索，而以后不断增加局部的开发。我们称之为非均匀变异，并将在本章的后面进行讨论。首先，我们讨论应用该新算子的测试问题。

6.1 测试例子

通常，对优化控制问题的求解，算法的设计和实现是困难的。极度宣扬的动态规划是一种可以用在各种有前后关系的问题里的数值技术，特别是优化控制[37]。该算法将问题拆解为规模和复杂性适度的子问题，但遇到了所谓的"维数灾难(the curse of dimensionality)"所带来的困难[33]。

优化控制问题的数值化处理是相当难的。一般用户可获得的某种数值动态优化程序其实是典型的固定式程序包的产品[50]，它们不使用特定的动态优化方法。这样，所用的程序就不能对 Hamilton、截断性条件(transversality conditions)等进行显式地使用。从另一

方面来说，如果它们确实使用专门的动态优化方法，对一个门外汉来说，将更难以处理。

另外，就作者所知，直到最近遗传算法才以一种系统的方法被用于优化控制问题[271,273]。我们相信先前的遗传算法对高精度要求问题是太无能为力了。本章将提出一个修正的遗传算法以增强其性能。并通过比较一些动态优化问题来展示所开发的系统的质量和应用性。随后，包含约束条件的这类优化问题的系统进一步被开发，这将在下一章（第 7.2 节）进行讨论。作为这些测试例子的参考，和本书所讨论的其他实算一样，我们使用求解这类问题的一个标准的计算包：带 MINOS 优化器的遗传算法模拟系统的学生版[150]。本书的其余部分以 GAMS 表示它们。

这里选择了三个简单的、常用于优化控制的时间离散优化控制模型来作为测试演化程序的问题：线性二次方问题（linear–quadratic problem）、收获问题（harvest problem）及离散化的推车问题（discretized push–cart problem）。我们将依次讨论它们。

6.1.1　线性二次方问题

第一个测试问题是一个一维的线性约束二次模型：

$$\min\ q\cdot x_N^2+\sum_{k=0}^{N-1}(s\cdot x_k^2+r\cdot u_k^2) \tag{6.1}$$

并服从

$$x_{k+1}=a\cdot x_k+b\cdot u_k\ ,\quad k=0,1,\ldots,N-1 \tag{6.2}$$

这里 x_0 给定，a,b,q,s,r 为给定常数，$x_k\in R$ 为系统的状态值，$u_k\in R$ 为系统的控制值。

服从式（6.2）的式（6.1）的最优性能值为：

$$J^*=K_0x_0^2 \tag{6.3}$$

这里 K_k 为 Riccati 方程的解：

$$K_k=s+ra^2K_{k+1}/(r+b^2K_{k+1}),\quad K_N=q \tag{6.4}$$

接下来求解服从式（6.2）的问题（6.1）的参数，如表 6.1 所示。在实算中，N 的值设定为 45。这是 GAMS 能获得可比拟的数值解的最大范围。

表 6.1　十次测试事例

事例	N	x_0	s	r	q	a	b
I	45	100	1	1	1	1	1
II	45	100	10	1	1	1	1
III	45	100	1000	1	1	1	1
IV	45	100	1	10	1	1	1
V	45	100	1	1000	1	1	1
VI	45	100	1	1	0	1	1
VII	45	100	1	1	1000	1	1
VIII	45	100	1	1	1	0.01	1
IX	45	100	1	1	1	1	0.01
X	45	100	1	1	1	1	100

6.1.2　收获问题

收获问题定义为：

$$\max\sum_{k=0}^{N-1}\sqrt{u_k} \tag{6.5}$$

并服从生长方程 $x_{k+1} = a \cdot x_k - u_k$ (6.6)

及一个等式约束 $x_0 = x_N$ (6.7)

这里，初始态值 x_0 给定，a 为一个常量，$x_k \in R$ 和 $u_k \in R^+$分别为状态值和非负控制值。

服从（6.6）和（6.7）式的问题（6.5）的优化值为：

$$J^* = \sqrt{\frac{x_0 \cdot (a^N - 1)^2}{a^{N-1} \cdot (a-1)}} \tag{6.8}$$

对上述问题（6.5）采用的参数为：$a = 1.1$，$x_0 = 100$ 及 $N = 2, 4, 10, 20, 45$。

6.1.3 推车问题

推车问题是求在给定时间（譬如一个单位）整个旅行距离的最大值 $x_1(N)$减去总消耗。系统是二阶的：

$$x_1(k+1) = x_2(k) \tag{6.9}$$

$$x_2(k+1) = 2x_2(k) - x_1(k) + \frac{1}{N^2}u(k) \tag{6.10}$$

欲求最大化的性能指数为：

$$x_1(N) - \frac{1}{2N}\sum_{k=0}^{N-1} u^2(k) \tag{6.11}$$

对此问题，式（6.11）的最优指数值为：

$$J^* = \frac{1}{3} - \frac{3N-1}{6N^2} - \frac{1}{2N^3}\sum_{k=0}^{N-1} k^2 \tag{6.12}$$

推车问题在不同的 N =5, 10, 15, 20, 25, 30, 35, 40, 45 下求解。注意不同的 N 对应于离散化周期（或者对等的连续问题）的数目，而不是优化范围假设为 1 的实际长度。

6.2 数值优化的演化程序

我们建立的数值优化演化程序是基于浮点表达及一些新的特殊处理的遗传算子；下面将依次对它们进行讨论。

6.2.1 浮点表达

在浮点表达里，每个染色体向量被编码成一个与解向量相同长度的浮点数向量。每个元素的初始选择都是在要求的域里进行的，算子被精心设计以保持这一约束条件（在二进制表达中不存在这样的问题，而算子的设计相当简单；我们不把它看成是一个缺点；相反，它提供另外的优点）。

这种方法的精度依赖于所用的机器，但总的来说远好于二进制表达。当然，我们可以通过引入更多的位来扩大二进制表达的精度，但正如前面章所讨论的，这会造成算法明显地变慢。

另外，浮点表达能够表达十分大的范围（或者未知的范围）。而对固定的二进制长度，二进制表达必须牺牲精度来增加范围的尺度。还有，浮点表达更容易被用来设计处理非常规约束的特殊的工具：这将在下一章中进行详细的讨论。

6.2.2　特殊算子

我们使用的算子和经典的算子不同，因为它工作在实型空间里。然而，出于直觉的相似性，我们将把它们分为标准类、变异类和杂交类。另外，一些算子不是一成不变的，即它们的行动依据群体的年龄。

1. 变异类：

- 均匀变异　和经典版本定义相似：如果 $x_i^t=\langle v_1,\dots,v_n\rangle$为一个染色体，那么每个元素 v_k 都有相同的机会经历变异。这一算子的单一应用结果是向量$\langle v_1,\dots,v_k',\dots,v_n\rangle$，其中 $1\le k\le n$，且 v_k' 取自对应参数 $domain_k$ 域中的一个随机值。
- 非均匀变异　是对应于系统的微调能力的算子。它定义如下：如果$s_v^t=\langle v_1,\dots,v_m\rangle$为一个染色体，且元素 v_k 被选择变异（v_k 的域为$[l_k,u_k]$），则结果为向量 $s_v^{t+1}=\langle v_1,\dots,v_k',\dots,v_m\rangle$，其中 $k\in\{1,\dots,n\}$，及：

$$v_k'=\begin{cases}v_k+\Delta(t,u_k-v_k) & \text{如果随机数为 0}\\ v_k+\Delta(t,u_k-v_k) & \text{如果随机数为 1}\end{cases}$$

这里，函数$\Delta(t,y)$返回$[0,y]$里的一个值，以使$\Delta(t,y)$靠近 0 的概率随 t 的增加而增加。这一性质使该算子在初始阶段均匀地搜索空间（当 t 很小时），而在后面阶段非常地局部化。我们使用了下面的函数：

$$\Delta(t,y)=y\cdot\left(1-r^{\left(1-\frac{t}{T}\right)^b}\right)$$

这里 r 为一个在[0, 1]里的随机数，T 为最大代数，b 为决定非均匀度的系统参数。图6.1 显示了两个被选时间的Δ值；该图清楚地表明了算子的行为。

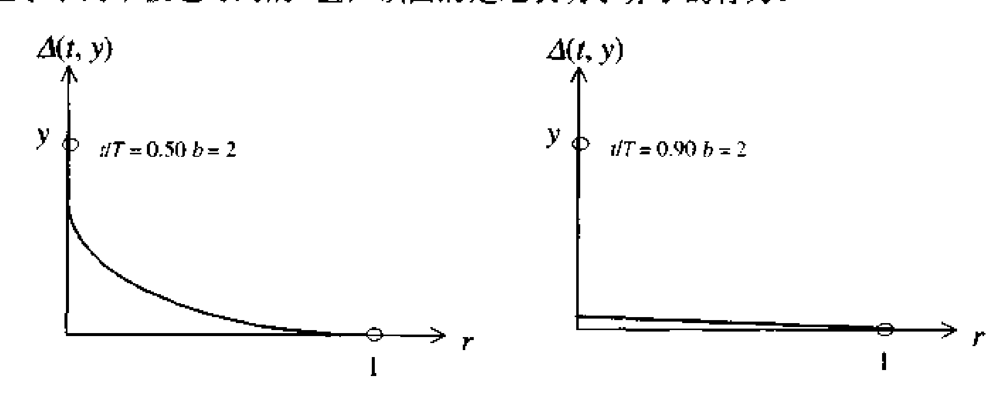


图 6.1　两个被选时间的Δ（t，y）值

除了使用标准的变异方法，我们还有一些新的机制：如，非均匀变异也应用在整个解向量，而不是它的单个成员，这引起整个向量在空间上一些轻微的滑动。

2. 杂交类:

- 简单杂交 按常用的方式定义，但对一给定染色体 x，只允许 v 之间的分裂点。
- 算术杂交(arithmetical crossover) 被定义为两个向量的线性组合：如果 s_v^t 和 s_w^t 被选择杂交，产生的后代是：$s_v^{t+1}=a\,s_w^t+(1-a)\,s_v^t$ 和 $s_w^{t+1}=a\,s_v^t+(1-a)\,s_w^t$。该算子可以使用的一个参数 a 或者是一个常数 (均匀算术杂交)，或者其值依赖于代龄（非均匀算术杂交)。

这里我们再次应用这些算子的新机制，例如算术杂交既可应用于两个向量的被选择元素，也可应用于整个向量。

6.3 实算和结果

在这一节，我们给出优化控制问题的演化程序结果，对所有测试问题，群体规模固定为 70，运行 40000 代。对每个测试事例，我们进行了三次随机的运行并报告最好结果：不管怎样，值得注意的是这样运行的标准偏差几乎小到可以忽略。向量$\langle u_0,\ldots,u_{N-1}\rangle$的初始化是随机的（但在要求的域里）。表 6.2，6.3 和 6.4 报告了伴随着某些代的中间结果所找到的最终值，例如，栏“10000”的值表示要运行 40000 代时，经过 10000 代后的阶段结果。值得注意的是，这样的值比只运行 10000 代时所获得的结果要差，这是因为一些遗传算子的性质造成的。在下一节，我们将把这些结果和从计算包 GAMS 得到的精确解做比较。

表 6.2 对线性二次问题(6.1) ~ (6.2)的演化程序结果

事例	代数							因子
	1	100	1000	10000	20000	30000	40000	
I	17904.4	3.87385	1.73682	1.61859	1.61817	1.61804	1.61804	10^4
II	13572.3	5.56187	1.35678	1.11451	1.09201	1.09162	1.09161	10^5
III	17024.8	2.89355	1.06954	1.00952	1.00124	1.00102	1.00100	10^7
IV	15082.1	8.74213	4.05532	3.71745	3.70811	3.70162	3.70160	10^4
V	5968.42	12.2782	2.69862	2.85524	2.87645	2.87571	2.87569	10^5
VI	17897.7	5.27447	2.09334	1.61863	1.61837	1.61805	1.61804	10^4
VII	2690258	18.6685	7.23567	1.73564	1.65413	1.61842	1.61804	10^4
VIII	123.942	72.1958	1.95783	1.00009	1.00005	1.00005	1.00005	10^4
IX	7.28165	4.32740	4.39091	4.42524	4.31021	4.31004	4.31004	10^5
X	9971341	148233	16081.0	1.48445	1.00040	1.00010	1.00010	10^4

注意，问题 (6.5) ~ (6.7)都有受约束的最终态。它不同于问题 (6.1) ~ (6.2)之处在于：不是每个随机初始化的正实数向量$\langle u_0,\ldots,u_{N-1}\rangle$都产生一个允许的序列 x_k（见条件(6.6)），使对给定的 a 和 x_0 有 $x_0=x_N$。在我们的演化程序里，产生一个随机序列 $u_0,\ldots,u_{N-2}$，并设 $u_{N-1}=a\,x_{N-1}-x_N$。对负的 u_{N-1}，我们放弃此序列，并重复初始化过程（下一章 7.2 节将详细讨论这个过程）。在复制过程中，会出现同样的困难。某个后代经过某种遗传操作后，不必一定要满足约束条件 $x_0=x_N$。在这种情况下，我们用公式 $u_{N-1}=a\,x_{N-1}-x_N$

替换后代向量 u 的最后成员。再有，如果 u_{N-1} 产生负值，则不把该后代引入到新群体中。

本章考虑的测试问题仅是包含非常规约束。约束优化控制问题的大致要点将在下一章进行讨论。

表 6.3 收获问题(6.5)~(6.7)的演化程序结果

N	代数						
	1	100	1000	10000	20000	30000	40000
2	6.3310	6.3317	6.3317	6.3317	6.3317	6.3317	6.331738
4	12.6848	12.7127	12.7206	12.7210	12.7210	12.7210	12.721038
8	25.4601	25.6772	25.9024	25.9057	25.9057	25.9057	25.905710
10	32.1981	32.5010	32.8152	32.8209	32.8209	32.8209	32.820943
20	65.3884	68.6257	73.1167	73.2372	73.2376	73.2376	73.237668
45	167.1348	251.3241	277.3990	279.0657	279.2612	279.2676	279.271421

表 6.4 推车问题(6.9)~(6.11)的演化程序结果

N	代数						
	1	100	1000	10000	20000	30000	40000
5	−3.008351	0.081197	0.119979	0.120000	0.120000	0.120000	0.120000
10	−5.668287	−0.011064	0.140195	0.142496	0.142500	0.142500	0.142500
15	−6.885241	−0.012345	0.142546	0.150338	0.150370	0.150370	0.150371
20	−7.477872	−0.126734	0.149953	0.154343	0.154375	0.154375	0.154377
25	−8.668933	−0.015673	0.143030	0.156775	0.156800	0.156800	0.156800
30	−12.257346	−0.194342	0.123045	0.158241	0.158421	0.158426	0.158426
35	−11.789546	−0.236753	0.110964	0.159307	0.159586	0.159592	0.159592
40	−10.985642	−0.235642	0.072378	0.160250	0.160466	0.160469	0.160469
45	−12.789345	−0.342671	0.072364	0.160913	0.161127	0.161152	0.161152

6.4 演化程序与其他方法

在这一节，我们将用精确解及从计算包 GAMS 所获得的结果来比较上述结果。

表 6.5 线性二次方问题解的比较

	精确解	演化程序		GAMS	
事例	数值	数值	D	数值	D
I	16180.3399	16180.3928	0.000%	16180.3399	0.000%
II	109160.7978	109161.0138	0.000%	109160.7978	0.000%
III	10009990.0200	10010041.3789	0.000%	10009990.0200	0.000%
IV	37015.6212	37016.0426	0.000%	37015.6212	0.000%
V	287569.3725	287569.4357	0.000%	287569.3725	0.000%
VI	16180.3399	16180.4065	0.000%	16180.3399	0.000%
VII	16180.3399	16180.3784	0.000%	16180.3399	0.000%
VIII	10000.5000	10000.5000	0.000%	10000.5000	0.000%
IX	431004.0987	431004.4182	0.000%	431004.0987	0.000%
X	10000.9999	10001.0038	0.000%	10000.9999	0.000%

6.4.1 线性二次方问题

对表 6.1 中指定的参数值，该问题的精确解已可从公式（6.3）和（6.4）获得。

为突出演化程序的性能和竞争性，用 GAMS 对同样的问题进行了求解。比较可以被认为是对演化程序不完全公平的，因为 GAMS 所用的搜索法特别适合于线性二次方问题。即问题 (6.1)~(6.2) 对该计算包必定是容易的事例。另一方面，如果对这些测试问题，演化程序被证明是有或差不多有竞争力的话，那么这就可表明它总体上的表现是令人满意的。表 6.5 概述了结果，栏 D 指相对误差的百分数。

如上可见，对线性二次方问题，GAMS 的执行结果是理想的。但是，对第二个测试问题的各子例，却不完全是这样。

6.4.2 收获问题

首先，请注意没有一个 GAMS 的解是和解析解完全相同的。解的差异随着表 6.6 中所示的优化规模一起增大，且对 $N>4$，系统找不到任何值。

这暴露出 GAMS 对优化问题的非凸性及变量数是敏感的。尽管向问题加入另外的约束（$u_{k+1}>0.1\cdot u_k$）来限制可行集使 GAMS 不会“迷路”[①]，但这并不能有多大的帮助（见“GAMS+”栏）。正如此栏显示的，优化套路充分，仍无机会从 GAMS 获得满意的解。

表 6.6 收获问题解的比较（符号“*”表示 GAMS 不能报告合理解）

N	精确解	GAMS		GAMS+		遗传算法	
	值	值	D	值	D	值	D
2	6.331738	4.3693	30.99%	6.3316	0.00%	6.3317	0.000%
4	12.721038	5.9050	53.58%	12.7210	0.00%	12.7210	0.000%
8	25.905710	*		18.8604	27.20%	25.9057	0.000%
10	32.820943	*		22.9416	30.10%	32.8209	0.000%
20	73.237681	*		*		73.2376	0.000%
45	279.275275	*		*		279.2714	0.001%

6.4.3 推车问题

对推车问题，GAMS 和演化程序都产生很好的结果（表 6.7）。但值得注意的是不同的搜索算法完成任务所花费的时间方面的变化关系。

表 6.7 推车问题解的比较

	精确解	GAMS		遗传算法	
N	值	值	D	值	D
5	0.120000	0.120000	0.000%	0.120000	0.000 %
10	0.142500	0.142500	0.000%	0.142500	0.000 %
15	0.150370	0.150370	0.000%	0.150370	0.000 %
20	0.154375	0.154375	0.000%	0.154375	0.000 %
25	0.156800	0.156800	0.000%	0.156800	0.000 %
30	0.158426	0.158426	0.000%	0.158426	0.000 %
35	0.159592	0.159592	0.000%	0.159592	0.000 %
40	0.160469	0.160469	0.000%	0.160469	0.000 %
45	0.161152	0.161152	0.000%	0.161152	0.000 %

对多数优化程序，算法收敛于最优解的时间需要量依赖于决策变量数。对动态规划，

① 从遗传算法的观点来看，这是“不公平的”，因为遗传算法没有这种帮助。

这种关系是指数型的“维数灾难”。对像GAMS这样的搜索方法，它通常“比线性差”。

表6.8报告了演化程序获得小数点后六位精度的解所需要的迭代数、为达精度所需的时间及40000代迭代耗费的总时间（对未知精确解，我们不能确定当前解的精度）。同时也给出GAMS所用的时间。注意，GAMS是在PC Zenith z-386/20上运行的，演化程序是在DEC-3100工作站上运行的。

很明显，演化程序比GAMS慢得多：所用的CPU时间的绝对值和使用的计算机都不同。不过，我们不去比较系统完成计算所需的时间，而是比较它们时间增长率随问题大小变化的关系。图6.2显示了演化程序和GAMS获得结果所需的时间增长率。

表6.8　演化程序和GAMS用于推车问题(6.9)~(6.11)的时间性能（获得小数点后第六位的精度所需迭代次数，完成上述迭代次数所需时间和完成所有40 000次迭代所需的时间）

N	所需迭代次数	所需时间（CPU秒）	40 000次迭代所需时间（CPU秒）	GAMS时间（CPU秒）
5	6234	65.4	328.9	31.5
10	10231	109.7	400.9	33.1
15	19256	230.8	459.8	36.6
20	19993	257.8	590.8	41.1
25	18804	301.3	640.4	47.7
30	22976	389.5	701.9	58.2
35	23768	413.6	779.5	68.0
40	25634	467.8	850.7	81.3
45	28756	615.9	936.3	95.9

这些图是能直观解释的：虽然演化程序一般来说较慢，但其接近于线性的增长率要远优于GAMS（后者至少是二次方增长率）。对线性二次方问题和收获问题，相似的结果也是成立的。

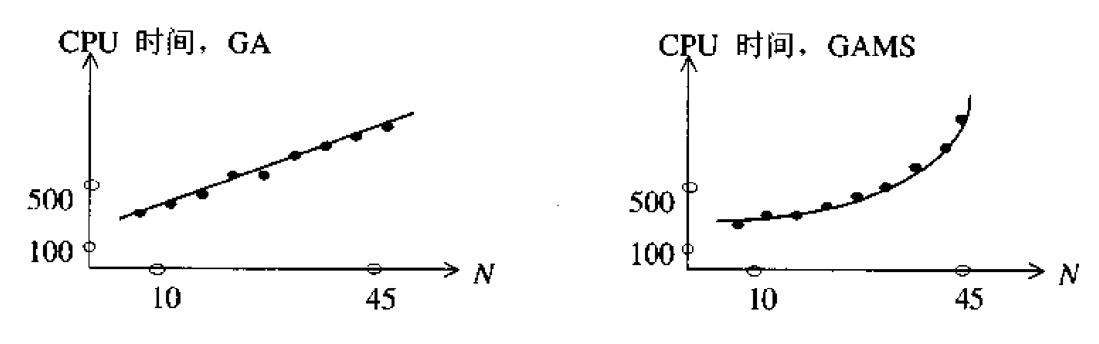


图6.2　CPU时间对问题规模（N）的函数

6.4.4 非均匀变异的优越性

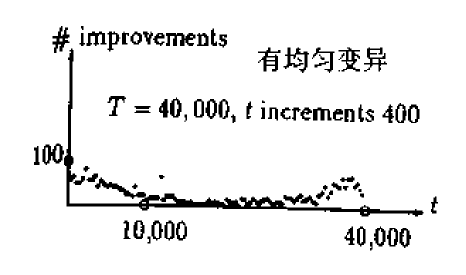
将精确解同另一种遗传算法——除了没有非均匀变异其他一样——所获得的解进行比较是有意义的。表6.9概述了结果；标以D的栏表示相对误差百分数。

使用了非均匀变异的遗传算法明显要胜于别的算法；增强的遗传算法误差很少大于十万分之几，而其他方法很难到达百分之一。而且，它还更快地收敛。

图6.3说明非均匀变异在演化过程中的效果。新的变异在群体生命的末尾可产生在群体中可观察到的改进数的较大增长。然而，在这个时刻之前改进数要小得多，再与实际的快速收敛合在一起清楚地说明：非均匀变异整体上是一种更好的搜索。

表6.9 线性二次动态控制问题解的比较

事例	精确解	带非均匀变异的遗传算法		不带非均匀变异的遗传算法	
	数值	数值	D	数值	D
I	16180.3399	16180.3939	0.000%	16234.3233	0.334%
II	109160.7978	109163.0278	0.000%	113807.2444	4.257%
III	10009990.0200	10010391.3989	0.004%	10128951.4515	1.188%
IV	37015.6212	37016.0806	0.001%	37035.5652	0.054%
V	287569.3725	287569.7389	0.000%	298214.4587	3.702%
VI	16180.3399	16180.6166	0.002%	16238.2135	0.358%
VII	16180.3399	16188.2394	0.048%	17278.8502	6.786%
VIII	10000.5000	10000.5000	0.000%	10000.5000	0.000%
IX	431004.0987	431004.4092	0.000%	431610.9771	0.141%
X	10000.9999	10001.0045	0.000%	10439.2695	4.380%



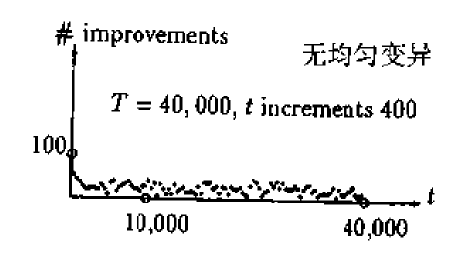


图6.3 线性约束二次动态控制问题实例I的改进数

6.5 结论

在本章，我们研究了一个新算子，即非均匀变异来改进遗传算法的局部调节能力。对三个被选择作为测试例子的离散时间优化控制问题，成功的实算结果是令人鼓舞的，因为解析解和数值解的靠近程度是令人满意的。另外，计算消耗是合理的。对40000代，在CRAY Y-MP上的CPU时间是几分钟，在DEC-3100工作站上最多是15分钟。

同时，对数值结果和一个基于搜索的计算包（GAMS）所获得的结果进行了比较。对所有的测试问题，演化程序和解析结果是可以比拟的，而GAMS对其中之一运行失败。所开发的演化程序展示了一些其他（基于梯度的）系统不总是具备的特点：

- 对演化程序，优化函数不必是连续的。而一些优化包完全不能接受这样的函数。
- 一些优化包是定义在或者全部或者没有命题之上的：用户不得不等待程序去完成。有时它不可能在早期阶段得到部分（或近似）解。演化程序给用户额外的灵活性，因为用户可以在运行时间内监控“搜索的状态”并作出适当的决定。而且，用户可以指定愿意支付的机时长短（时间越长，解越精确）。
- 演化程序的计算复杂性呈线性比率增加；而多数其他搜索方法对优化水平十分敏感。照例，我们可以用并行实现来很容易地改进系统的性能；而这对其他优化方法经常是困难的。

最近，在演化算法领域有许多有意义的发展以增强它们的局部微调能力。其中包括δ

算法(Delta Coding algorithm)、动态参数编码(Dynamic Parameter Encoding)、ARGOT 策略、IRM 策略(Immune recruitment mechanism)、演化规划扩展(Evolutionary programming)、粒性演化(granularity evolution)及间隔遗传算法(interval genetic algorithms)。在讨论完 GENOCOP 系统系列（第 7 章）及演化策略（第 8 章）之后，第 8 章的最后一节对有关尝试进行了描述。

还有一些其他方面的尝试，或多或少是针对局部微调的。包括 Arabas 等的研究[11]。其中为演化策略引入了自适应中间体及均匀杂交（见第 8 章）。另外，Hinterding[184]实算了相对位变异的基因块（对应于变量的）变异。从这一角度出发，作者分析了编码（灰码对二进制正码）、粒性及基因变异频率的重要性。

Srinivas 和 Patnaik[368]同样报告了有趣的实算结果，他们实算了杂交和变异的自适应概率以保持群体的多样性并维持算法的收敛能力。在这种方法里，这些算子的概率根据解的适应值而变化："好"的解被保护；"差"的解被破坏。更精确地说，

$$p_c = \begin{cases} k_1 \cdot (f_{\max} - f')/(f_{\max} - \overline{f}) & 当\ f' \le \overline{f} \\ k_3 & 当\ f' > \overline{f} \end{cases}$$

及

$$p_m = \begin{cases} k_2 \cdot (f_{\max} - f')/(f_{\max} - \overline{f}) & 当\ f \le \overline{f} \\ k_4 & 当\ f > \overline{f} \end{cases}$$

这里 k_1 和 k_2 是正的常数（不大于 1），f_{max} 和 $\overline{f}$ 分别表示当前群体中适应值函数 f 的最大值和平均值，f 表示给定解的适应函数值，f' 是较大值（选择杂交的两个解之一）。注意：

(1) $f_{max} - \overline{f}$ 的值对上述公式是基本的；它对度量算法的收敛性也非常重要；

(2) p_c 和 p_m 对最大适应值的解为零；

(3) 对 $f = \overline{f}$ 的解，$p_c = k_1$ 及 $p_m = k_2$；

(4) 对一个平均值之下的解，$p_c = k_3$ 及 $p_m = k_4$。

有关 k_1, k_2, k_3 及 k_4 值的选择和实算结果见[368]。

第 7 章　处理约束技巧

一般的非线性规划问题 *NLP* (nonlinear programming, NLP) 的提法是找到 $\boldsymbol{x}$ 以优化 $f(\boldsymbol{x})$，其中 $\boldsymbol{x}=(x_1,\dots,x_q)\in R^q$，并使其符合 $p\geq 0$ 个等式：

$$c_i(\boldsymbol{x})=0,\qquad i=0,\dots,p$$

及 $m-p\geq 0$ 个不等式：

$$c_i(\boldsymbol{x})\leq 0,\qquad i=p+1,\dots,m$$

然而还没有一种现成的方法来确定一般的非线性规划问题的全局最大（或最小）值。而只有当目标函数 f 及约束 c_i 满足某种性质时，全局最优有时才能被找到。已开发出来的几个算法是针对非约束问题（直接搜索法、梯度法）和约束问题（这些算法通常被分成直接法和间接法）。间接法通过从原始问题中抽取一个或多个线性问题来获得解，而直接法试图确定相继的搜索点，这通常是将原始问题转变为非约束问题，再用一些修正的梯度法来求解[387]。虽然近些年全局优化的研究和进展相当活跃[114]，认同的看法是：对一般非线性问题 *NLP* 尚未见到有效的解决方案，正如[172]中描述的：

> "找到一个能解决任一类非线性模型的通用 *NLP* 程序是不现实的。退而求其次，你应该试着选择一种适合你正在着手解决的问题的程序。如果你的问题对除了'一般'之外的任何策略都不适应的话，或者你一定要全局最优解（除你没有机会遇到多个局部最优解的时候），你应当有不得不使用穷举搜索的思想准备，即你遇到了很棘手的问题。"

还有许多其他与传统优化方法关联的难题。如多数已经提出来的优化方法其工作范围是局域性的，它们依赖于导数的存在，而对不连续的、大范围多峰性或者带噪声的搜索空间，它们缺少足够的鲁棒性。因此，研究其他启发性方法是非常重要的，这对许多真实的现实问题可能很有用。

本章将讨论几个已开发出来的与非线性规划问题有关的方法。我们将从对 GENOCOP 系统的描述开始，它是被开发来解凸形空间的。在后面部分，我们将综述其他解非线性规划问题的演化方法，并展示两个系统：GENOCOP II 和 GENOCOP III。

7.1 一个演化程序：GENOCOP 系统

许多研究者[78,408]研究了基于浮点表达的遗传算法。但是，他们考虑的优化问题都是定义在搜索空间 $D\subseteq R^q$ 里的，这里 $D=\prod_{k=1}^{q}\langle l_k,r_k\rangle$，即每个变量 x_k 被限制在一给定的区间 $\langle l_k,r_k\rangle$（$1\leq k\leq q$）里。看起来，将其他的约束包括到考虑的问题中是很重要的；正

如[66]中所述的：

> "稍微进行一下观察和思考，就会得出现实的所有优化问题实际上都是带约束问题。假定用一个表达式来表达正进行几种化学反应容器的产量，希望其产量最大，这就必须额外考虑反应物和生成物的物料平衡计算、控制进出反应容器物料的流动定律及其他条件。所用这些都是待优化函数变量的附加约束。"

在一个约束优化问题里，R^q 里解集合的几何形状对在尝试求解问题中可能遇到的难度也许是最关键的特性[66]。仅仅只对一类特殊形状的集合 — 凸集 — 已经发展了有份量的理论。

在本节里，我们主要关心下面的优化问题：

$$\text{优化：}\quad f(x_1,\ldots,x_q)\in R$$

这里$(x_1,\ldots,x_q)\in D\subseteq R^q$，$D$ 为凸集，且被变量范围（$l_k\le x_k\le r_k$，其中 $k=1,\ldots,q$）及约束集合 C 所定义。从集合 D 的凸性可以得到：对搜索空间上的每个点 $(x_1,\ldots,x_q)\in D$，存在变量 x_k（$1\le k\le q$）的可行范围 $\langle left(k), right(k)\rangle$，而其他变量 x_i（$i=1,\ldots,k-1,\ k+1,\ldots,q$）保持固定。换句话说，对给定的 $(x_1,\ldots,x_k,\ldots,x_q)\in D$：

$$y\in\langle left(k), right(k)\rangle，\text{当且仅当}\ (x_1,\ldots,x_{k-1},y,x_{k+1},\ldots,x_q)\in D$$

这里，所有的 x_i（i=1,...,k-1,k+1,...,q）保持恒定。我们同样可以假设区间$\langle left(k), right(k)\rangle$能有效地计算。

例如，如果 $D\subseteq R^2$ 被定义为：

$$-3\le x_1\le 3$$
$$0\le x_2\le 8$$
$$x_1^2\le x_2\le x_1+4$$

那么，对一个给定点$(2,5)\in D$：

$$left(1)=1,\quad right(1)=\sqrt{5},$$
$$left(2)=4,\quad right(2)=6$$

这就意味着，向量(2,5)的第一个元素可以在 $1\sim\sqrt{5}$ 之间变化 （此时 $x_2=5$ 保持固定）而第二个向量元素可以在 4~6 之间变化（此时 $x_1=2$ 保持固定）。

当然，如果约束集合 C 是空集，那么搜索空间 $D=\prod_{k=1}^{q}\langle l_k, r_k\rangle$ 为凸；另外对 $k=1,\ldots,q$，有 $left(k)=l_k$，$right(k)=r_k$。

上述性质是所有变异算子的基础：如果变量 x_k 被变异，变异范围为$\langle left(k), right(k)\rangle$；因此变异后代总是可行的。

凸搜索空间的另一个性质是：对解空间 D 里的任何两个点 $\boldsymbol{x}_1$，$\boldsymbol{x}_2$ 的线性组合：

$$a\boldsymbol{x}_1+(1-a)\boldsymbol{x}_2$$

同样是 D 中的一个点，其中 $a\in[0,1]$。此性质对算术杂交的实现是很重要的。

考虑定义在凸域上一类特殊的优化问题，这些问题可以用如下公式表达。

优化一个函数$f(x_1,x_2,...,x_q)$，使其服从下面的线性约束集合：

(1) 域约束：$l_i \le x_i \le u_i$ 对 i=1,2,..., q。记作$\boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$，其中$\boldsymbol{l} = \langle l_1, ..., l_q \rangle$，$\boldsymbol{u} = \langle u_1, ...,u_q \rangle$，$\boldsymbol{x} = \langle x_1, ..., x_q \rangle$。

(2) 等式约束：$A\boldsymbol{x} = \boldsymbol{b}$，$\boldsymbol{x} = \langle x_1, ..., x_q \rangle$，$A=(a_{ij})$，$\boldsymbol{b} = \langle b_1,...,b_p \rangle$，其中 $1 \le i \le p$ 及 $1 \le j \le q$（p 为等式数目）。

(3) 不等式约束：$C\boldsymbol{x} \le \boldsymbol{d}$，$\boldsymbol{x} = \langle x_1, ..., x_q \rangle$，$C=(c_{ij})$，$\boldsymbol{d} = \langle d_1,...,d_m \rangle$，$1 \le i \le m$，$1 \le j \le q$（$m$ 为不等式数目）。

上述公式的一般性足以处理一大类标准的带有线性约束和任何目标函数的运筹学优化问题。后面用到的非线性运输问题例子就是许多这类问题中的一个。

已开发的系统[GENOCOP（约束问题数值优化的遗传算法）：GEnetic algorithm for Numerical Optimization for COnstrained Problems]提供了处理约束的一种方法，该方法既有一般性又有问题的依赖性。它揉合了前面方法中出现的一些思想，但总的来说是较新的内容。该方法中包含的主要思想在于：（1）消除约束集中出现的等式；（2）对特殊遗传算子精心设计，它保证让所有的染色体都在约束空间里。这样做对线性约束非常有效，当然我们并不是自称这些结果可以很容易地延伸到非线性约束。只是线性约束包含许多有趣的优化问题。

在一些优化方法中，如线性规划，等式约束很受欢迎，因为如果最优解存在，它总是位于凸集的表面。不等式可通过附加松弛变量转变成等式，解的方法总是沿着表面从一个顶点转移到另一个顶点。

相反，对随机产生解的方法，这些等式约束却是令人麻烦的。在 GENOCOP 里，它们一开始就和相等数量的问题变量一起被消去；这样做就移去了这一部分搜索空间。留下的线性不等式形式的约束构成了寻找解时必须进行搜索的凸集。搜索空间的凸性确保了解的线性组合无需检验约束就产生新的解 — 贯穿整个方法使用的性质。不等式可以用来产生任何给定变量的边界：这样的边界是动态的，因为它依赖于其他变量的值，并能被有效地计算出。

假定等式约束集以矩阵形式表达：

$$A\boldsymbol{x} = \boldsymbol{b}$$

假定有p个独立的方程（它们是容易验证的），即有p个变量x_{i_1}，$x_{i_2},...,x_{i_p}$（$\{i_1,...,i_p\} \subseteq \{1,2,...,q\}$）可以由其他变量来确定。因此它们可从问题中消去，如下所示。

我们可以将数组A纵向分割成两个数组A_1和A_2，使矩阵A的j列属于A_1，当且仅当$j \in \{i_1,...,i_p\}$。这样A_1^{-1}是存在的。类似地，分割矩阵C与向量$\boldsymbol{x}$，$\boldsymbol{l}$，$\boldsymbol{u}$（即$\boldsymbol{x}^1 = \langle x_{i_1}, x_{i_2},...,x_{i_p} \rangle$，$\boldsymbol{l}_1 = \langle l_{i_1}, l_{i_2},...,l_{i_p} \rangle$及$\boldsymbol{u}_1 = \langle u_{i_1}, u_{i_2},...,u_{i_p} \rangle$）。那么

$$A_1\boldsymbol{x}^1 + A_2\boldsymbol{x}^2 = \boldsymbol{b}$$

可以容易地理解为：

$$\boldsymbol{x}^1 = A_1^{-1}\boldsymbol{b} - A_1^{-1}A_2\boldsymbol{x}^2$$

使用上述规则，我们可以用其余变量的线性组合替换变量 x_{i_1} ， $x_{i_2}, \ldots, x_{i_p}$ 。但是，每个变量 x_{i_j} （$j = 1,2, \ldots, p$） 受到额外的域的约束：$l_{i_j} \leq x_{i_j} \leq u_{i_j}$ 。消去所有的变量 x_{i_j} ，我们即得到新的不等式集：

$$\boldsymbol{l}_1 \leq A_1^{-1}\boldsymbol{b} - A_1^{-1}A_2\boldsymbol{x}^2 \leq \boldsymbol{u}_1$$

它被加到原来的不等式集中。

原来的不等式集：　　$C\boldsymbol{x} \leq \boldsymbol{d}$

- 于是就可以被表达成：　　$C_1\boldsymbol{x}^1 + C_2\boldsymbol{x}^2 \leq \boldsymbol{d}$

进一步转变为：　　$C_1(A_1^{-1}\boldsymbol{b} - A_1^{-1}A_2\boldsymbol{x}^2) + C_2\boldsymbol{x}^2 \leq \boldsymbol{d}$

消去 p 个变量 x_{i_1} ， $x_{i_2}, \ldots, x_{i_p}$ 后，最终的约束集由下面的不等式组成：

- 原来的约束区间：$\boldsymbol{l}_2 \leq \boldsymbol{x}^2 \leq \boldsymbol{u}_2$；
- 新的不等式：$\boldsymbol{l}_1 \leq A_1^{-1}\boldsymbol{b} - A_1^{-1}A_2\boldsymbol{x}^2 \leq \boldsymbol{u}_1$；
- 原来的不等式移去 $\boldsymbol{x}^1$ 变量后：$(C_2 - C_1A_1^{-1}A_2)\boldsymbol{x}^2 \leq \boldsymbol{d} - C_1A_1^{-1}\boldsymbol{b}$。

7.1.1 一个例子

让我们从一个例子开始，假定优化的是有六个变量的函数：

$$f(x_1, x_2, x_3, x_4, x_5, x_6)$$

并服从下面的约束：

$$2x_1 + x_2 + x_3 = 6$$
$$x_3 + x_5 - 3x_6 = 10$$
$$x_1 + 4x_4 = 3$$
$$x_2 + x_5 \leq 120$$
$$-40 \leq x_1 \leq 20,\quad 50 \leq x_2 \leq 75$$
$$0 \leq x_3 \leq 10,\quad 5 \leq x_4 \leq 15$$
$$0 \leq x_5 \leq 20,\quad -5 \leq x_6 \leq 5$$

我们可以利用存在的三个独立等式将三个变量表达成其余三个变量的函数：

$$x_1 = 3 - 4x_4$$
$$x_2 = -10 + 8x_4 + x_5 - 3x_6$$
$$x_3 = 10 - x_5 + 3x_6$$

这样，原来的约束问题缩减为三个变量 x_4, x_5 和 x_6 的函数优化问题：

$$g(x_4, x_5, x_6) = f(3-4x_4, -10+8x_4+x_5-3x_6, 10-x_5+3x_6, x_4, x_5, x_6)$$

并服从下面约束（只有不等式）：

$$-10 + 8x_4 + 2x_5 - 3x_6 \leq 120 \qquad \text{（原来的 } x_2 + x_5 \leq 120\text{）}$$
$$-40 \leq 3 - 4x_4 \leq 20 \qquad \text{（原来的 } -40 \leq x_1 \leq 20\text{）}$$
$$50 \leq -10 + 8x_4 + x_5 - 3x_6 \leq 75 \qquad \text{（原来的 } 50 \leq x_2 \leq 75\text{）}$$
$$0 \leq 10 - x_5 + 3x_6 \leq 10 \qquad \text{（原来的 } 0 \leq x_3 \leq 10\text{）}$$
$$5 \leq x_4 \leq 15,\quad 0 \leq x_5 \leq 20,\quad -5 \leq x_6 \leq 5。$$

更进一步简化：第 2 和第 5 个不等式可以变成一个：

$$5 \le x_4 \le 10.75$$

这些转换即完成了算法的第一步：等式的消除。导出的搜索空间当然是凸的。如前所述，根据搜索空间的凸性，对每一个可行点(x_1, x_2, x_3)，存在变量x_k（$1 \le k \le 3$）的可行范围$\langle left(k), right(k)\rangle$，此时其余的两个变量固定。例如，上述定义的可行空间，对一个可行点$(x_4, x_5, x_6) = (10,8,2)$：

$$left(1) = 7.25, \quad right(1) = 10.375,$$

$$left(2) = 6, \quad right(2) = 11,$$

$$left(3) = 1, \quad right(3) = 2.666,$$

［*left*(1)和 *right*(1)为向量(10,8,2)第 1 个元素即变量 x_4 的区间，余此类推］。这就意味着向量(10,8,2)的第 1 个元素可以在 7.25 到 10.375 之间变化 （其中 $x_5 = 8$ 和 $x_6 = 2$ 保持固定），该向量的第 2 个元素可以在 6 到 11 之间变化（此时 $x_4 = 10$ 及 $x_6 = 2$ 保持固定），该向量的第 3 个元素可以在 1 到 2.666 变化之间（此时 $x_4 = 10$ 及 $x_5 = 8$ 保持固定）。

GENOCOP 系统试图通过在可行区域取样来设置初始（可行）解。如果一些预定义数目的试探解不成功，系统将向用户提示调整可行初始点。初始群体由这些初始点（自动生成的或由用户提供的）的恒等复制组成。

GENOCOP 系统中有几个算子被证明对许多测试问题是有用的。我们将在下面部分依次对它们进行讨论。

7.1.2 算子

这一小节将描述用在 GENOCOP 系统修正版本中的基于浮点表达的六个遗传算子。前三个是一元算子（变异类），其余三个是二元算子（各种类型的杂交） 下面依次对它们进行讨论。

1. 均匀变异

均匀变异算子需要单个的亲体 $\boldsymbol{x}$ 产生单个的子代 $\boldsymbol{x}'$。算子选择向量 $\boldsymbol{x} = (x_1, \ldots, x_k, \ldots, x_q)$中一个随机元素 $k \in (1, \ldots, q)$并产生 $\boldsymbol{x}' = (x_1, \ldots, x_k', \ldots, x_q)$，这里 x_k' 为区域$\langle left(k), right(k)\rangle$里的一个随机值（均匀概率分布），

此算子在演化过程的早期起着重要的作用，因为解被允许在搜索空间里自由移动。特别是算子在对初始群体由同一可行点的多重复制组成的情况下是非常必要的。这种情况经常发生在需用户指定过程开始点的约束优化问题中。而且这样的单个开始点（撇开其缺点不说）有很强的优势：它可以被发展成一个迭代过程，其中的下一个迭代开始于先前迭代的最好点。这种技术可用在处理空间是非凸的非线性约束系统（GENOCOP II）。

另外，在演化过程的后期，该算子允许在搜索过程中离开局部最优解进行可能的移动，以便搜索更好点。

2. 边界变异

边界变异算子同样需要单个的亲体 $\boldsymbol{x}$ 产生单个的子代 $\boldsymbol{x}'$。该算子为均匀变异的一个

变种，其中的 x_k' 以相同的概率要么取 $left(k)$，要么取 $right(k)$。

算子被构造用于这样的优化问题，其最优解位于或靠近可行搜索空间边界。因此如果约束集 C 为空，变量的边界就十分宽，算子就会令人厌烦。但对存在约束的问题，它被证明是十分有用的。可用一个简单的例子来说明此算子的使用。例子是一个线性规划问题；在这种情况下，我们知道全局最优解位于搜索空间的边界上。

例 7.1

考虑下面的测试案例[387]：

最大化 $f(x_1, x_2) = 4x_1 + 3x_2$

并服从下面约束：

$$2x_1 + 3x_2 \le 6$$
$$-3x_1 + 2x_2 \le 3$$
$$2x_1 + x_2 \le 4$$
$$0 \le x_i \le 2, \quad i=1,2$$

已知全局最优解为 $(x_1, x_2) = (1.5, 1.0)$，　且 $f(1.5, 1.0) = 9.0$。

为确定算子在优化上述问题中的用途，十次运行的实算使用了所有的功能算子而另外十次没有使用边界变异。带边界变异的系统在十次实算中很容易地发现最优解，平均在 32 代之内。而没有该算子的实算甚至在 100 代才找到最好点（在十次运行中）。最好点为 $\boldsymbol{x} = (1.501, 0.997)$ 及 $f(\boldsymbol{x}) = 8.996$ ，最差点为 $\boldsymbol{x} = (1.576, 0.847)$ 及 $f(\boldsymbol{x}) = 8.803$。

3. 非均匀变异

这是用以对系统进行微调的一元算子。它被定义为，对一个亲体 $\boldsymbol{x}$，如果元素 x_k 被选择变异，　结果是 $\boldsymbol{x}' = \langle x_1, \ldots, x_k', \ldots, x_q \rangle$，　这里

$$x_k' = \begin{cases} x_k + \Delta(t, right(k) - x_k) & \text{如果随机二进制数为 0} \\ x_k - \Delta(t, x_k - left(k)) & \text{如果随机二进制数为 1} \end{cases}$$

函数 $\Delta(t, y)$ 返回一在区间 $[0, y]$ 里的值。于是随着 t 增加（t 为代数），$\Delta(t, y)$ 靠近 0 的概率就会增加。这种性质使算子初期可均匀地搜索空间（当 t 很小时），而在后期则具有局部性。我们使用下面的函数：

$$\Delta(t, y) = y \cdot r \cdot \left(1 - \frac{t}{T}\right)^b$$

其中 r 为区间 $[0, 1]$ 里的一个随机数，　T 为最大代数，b 为确定非均匀度的系统参数。

4. 算术杂交

此二元算子被定义为两个向量的组合：如果 x_1 和 x_2 被杂交，最终的后代为：

$$x_1' = a \cdot x_1 + (1-a) \cdot x_2 \text{ 及 } x_2' = a \cdot x_2 + (1-a) \cdot x_1$$

算子使用了一个随机值 $a \in [0, 1]$，用它总可以保证闭合（$x_1', x_2' \in D$）。该杂交被称为保证平均杂交(guaranteed average crossover)[77]（当 $a = 1/2$）；中间体杂交(intermediate crossover)[18]；线性杂交(linear crossover)[408]及算术杂交(arithmetical crossover)[268,269]。

算术杂交的重要性可以用下面的例子阐明。

例 7.2

考虑下面的问题[114]：

最小化 $f(x_1, x_2, x_3, x_4, x_5) = -5\sin(x_1)\sin(x_2)\sin(x_3)\sin(x_4)\sin(x_5) + -\sin(5x_1)\sin(5x_2)\sin(5x_3)\sin(5x_4)\sin(5x_5)$

其中 $0 \le x_i \le \pi$，$1 \le i \le 5$。

已知全局最优解为$(x_1, x_2, x_3, x_4, x_5) = (\pi/2, \pi/2, \pi/2, \pi/2, \pi/2)$及$f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$。

看来没有算术杂交的系统收敛得较慢。经过 50 代后（10 次运行结果），最好点的平均值为–5.9814，经过 100 代后的最好点的平均值为–5.9966。而对带算术杂交的系统，这些平均值分别为–5.9930 和–5.9996。

而且，十次运行获得的结果显示出有意义的模式：带算术杂交时系统更稳定，最好解的标准偏差低得多。

5. 简单杂交

此二元算子被定义为：如果 $x_1 = (x_1, \dots x_q)$ 和 $x_2 = (y_1, \dots, y_q)$ 被选择在第 k 位置后杂交，产生的后代是：

$$x_1' = (x_1, \dots, x_k, y_{k+1}, \dots, y_q) \text{ 和 } x_2' = (y_1, \dots, y_k, x_{k+1}, \dots, x_q)$$

这样的算子可能产生超出域 D 以外的后代。为避免这样，我们使用了凸空间表达性质。即存在 $a \in [0,1]$ 使下面两个后代是可行的。

$$x_1' = \langle x_1, \dots, x_k, y_{k+1} \cdot a + x_{k+1} \cdot (1-a), \dots, y_q \cdot a + x_q \cdot (1-a) \rangle \text{和}$$
$$x_2' = \langle y_1, \dots, y_k, x_{k+1} \cdot a + y_{k+1} \cdot (1-a), \dots, x_q \cdot a + y_q \cdot (1-a) \rangle$$

剩下要回答的唯一问题是如何找到最大的 a 以获得最大可能的信息交换。最简单的方法是从 $a=1$ 开始，如果至少有一个后代不属于 D，则用某个常数$\dfrac{1}{\rho}$来减少 a。用 ρ 来向 $a=0$ 方向调整，且两个后代都在 D 里，让其父代认同（意即可行点）。这种最大程度减小的必要性总体上是很小的，且在群体生命期间减小得很快。

看起来，简单杂交的优点和算术杂交的优点是相同的（实算中我们用了测试实例#6；见 7.1.3）。结果显示：没有简单杂交的系统甚至比没有算术杂交的系统更不稳定；在这种情况下，十次运行所获得的最好解的标准偏差要高得多。而且 100 代所获得的最差解(–5.9547)比使用所有算子的方法所获得的最差解的值（–5.9982）或者不使用算术杂交的方法所获得的最差解（–5.9919）更差。

6. 启发式杂交

此算子[408]为一独特的杂交，出于下面的理由：（1）它使用了目标函数值以确定搜索方向；（2）它只生成一个后代；（3）它可能根本不产生后代。

此算子从两个父代 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 中根据下面规则，产生单个后代 $\boldsymbol{x}_3$：

$$\boldsymbol{x}_3 = r \cdot (\boldsymbol{x}_2 - \boldsymbol{x}_1) + \boldsymbol{x}_2$$

这里 r 为在 0 和 1 之间的一随机数，亲体 $\boldsymbol{x}_2$ 不比 $\boldsymbol{x}_1$ 差，即对求最大问题，$f(\boldsymbol{x}_2) \ge f(\boldsymbol{x}_1)$；

对求最小问题 $f(\mathbf{x}_2) \le f(\mathbf{x}_1)$。

此算子有可能产生不可行解向量，在这种情况下，产生另一个随机数 r 及另一个后代。如果经过 w 次尝试后，没有发现新的服从约束的解，算子终止并不再产生后代。

看起来，启发式杂交更有利于发现精确解；其主要职能是：（1）微调；（2）朝一个最有希望的方向搜索。

7.1.3 测试 GENOCOP

为了评价 GENOCOP 方法，我们精心挑选了一套测试问题来测试算法的执行效果，并已被证明在实用中是成功的。下面将讨论八个测试实例，包括带几个线性约束的二次方、非线性及不连续函数。

系统所有运行都是在 SUN SPARC station 2 上执行的。所有的实算使用了下面的参数：

群体规模 $pop_size = 70$，$k = 28$（每代中的亲体数，分级阶段），非均匀变异的系数 $b = 2$。

（GENOCOP 系统的第三版可以从网址 ftp.uncc.edu 目录 coe/evol 中的 GENOCOP3.0.tar.z 文件获得）。对每个测试实例，运行 GENOCOP 十次。对所有问题，代数 T 为 500 或者为 1000（除了实例#6 运行了 10000 代）。下面部分报告了 GENOCOP 系统的八个测试实例和结果。

(1) 实例#1

问题[114]为：

最小化 $f(\mathbf{x}, y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$

服从约束：　$6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \le 6.5$，　$10x_1 + 10x_3 + y \le 20$

$0 \le x_i \le 1$，　$0 \le y$。

全局最优解为 $(\mathbf{x}^*, y^*) = (0,1,0,1,1,20)$，及 $f(\mathbf{x}^*, y^*) = -213$。GENOCOP 在十次运行中找到的解非常靠近最优解：找到的一个典型最优点为：

(0.000000, 1.000000, 0.000000, 0.999999, 1.000000, 20.000000)

相应的目标函数值等于−213.0。1000 次迭代的单次运行消耗 CPU 时间 21 秒。

(2) 实例#2

问题[186]为：

$$\text{最小化 } f(\mathbf{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + x_2 + \ldots + x_{10}}\right)$$

服从约束：

$$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2, \quad x_4 + 2x_5 + x_6 + x_7 = 1,$$

$$x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1, \quad x_i \ge 0.000001 \quad (i=1,\ldots,10)$$

其中

$c_1 = -6.089$；　$c_2 = -17.164$；$c_3 = -34.054$；$c_4 = -5.914$；　$c_5 = -24.721$；

$c_6=-14.986$； $c_7=-24.100$； $c_8=-10.708$； $c_9=-26.662$； $c_{10}=-22.179$；

此例子中的最知名的解[186]为：

$$x^*=(.01773548,\ .08200180,\ .8825646, .0007233256, .4907851,\\ .01727298,\ .007765639, .01984929, .05269826)$$

及 $f(x)=-47.707579$。GENOCOP 在所有十次运行中找到比上述解更好的解；找到的最好解为：

$$x^*=(.04034785, .15386976, .77497089, .00167479, .48468539,\\ .00068965, .02826479, .01849179, .03849563, .10128126)$$

相应的目标函数值等于 –47.760765。1000 次迭代的单次运行耗费 CPU 时间 56 秒。

(3) 实例#3

问题[114]为求最小：

$$f(\boldsymbol{x},\boldsymbol{y})=5x_1+5x_2+5x_3+5x_4-5\sum_{i=1}^{4}x_i^2-\sum_{i=1}^{9}y_i$$

服从约束：

$$2x_1+2x_2+y_6+y_7\le 10,\quad 2x_1+2x_3+y_6+y_8\le 10,$$
$$2x_2+2x_3+y_7+y_8\le 10,\quad -8x_1+y_6\le 0,$$
$$-8x_2+y_7\le 0,\quad -8x_3+y_8\le 0,$$
$$-2x_4-y_1+y_6\le 0,\quad -2y_2-y_3+y_7\le 0,$$
$$-2y_4-y_5+y_8\le 0,\quad 0\le x_i\le 1,\quad i=1,2,3,4$$
$$0\le y_i\le 1,\quad i=1,2,3,4,5,9,\quad 0\le y_i,\quad i=6,7,8$$

全局最优解为$(\boldsymbol{x}^*,\boldsymbol{y}^*)=(1,1,1,1,1,1,1,1,1,3,3,3,1)$， $f(\boldsymbol{x}^*,\boldsymbol{y}^*)=-15$。GENOCOP 在十次运行中均找到最优解：找到的一个典型最优点为：

(1.000000, 1.000000, 1.000000, 1.000000, 0.999995, 1.000000, 0.999999,
1.000000, 1.000000, 2.999984, 2.999995, 2.999995, 0.999999)

相应的目标函数值等于-14.999965。1000 次迭代的单次运行消耗 CPU 时间 41 秒。

(4) 实例#4

问题[113]为求最大：

$$f(\boldsymbol{x})=\frac{3x_1+x_2-2x_3+0.8}{2x_1-x_2+x_3}+\frac{4x_1-2x_2+x_3}{7x_1+3x_2-x_3}$$

服从约束：

$$x_1+x_2-x_3\le 1,\quad -x_1+x_2-x_3\le -1$$
$$12x_1+5x_2+12x_3\le 34.8,\quad 12x_1+12x_2+7x_3\le 29.1$$
$$-6x_1+x_2+x_3\le -4.1,\quad 0\le x_i,\quad i=1,2,3$$

全局最优解为 $\boldsymbol{x}^*=(1,0,0)$，$f(\boldsymbol{x}^*)=2.471428$。GENOCOP 在所有的十次运行中均找到最优解；一个 500 代的单次运行消耗 CPU 时间 10 秒。

(5) 实例#5

问题[114]为：最小化 $f(\boldsymbol{x})=x_1^{0.6}+x_2^{0.6}-6x_1-4x_3+3x_4$

服从约束：

$$-3x_1 + x_2 - 3x_3 = 0,\qquad x_1 + 2x_3 \le 4$$
$$x_2 + 2x_4 \le 4,\qquad x_1 \le 3$$
$$x_4 \le 1,\qquad 0 \le x_i,\quad i=1,2,3,4$$

已知全局最优解为 $x^* = (4/3, 4, 0, 0)$，$f(x^*) = -4.5142$。GENOCOP 在所有的十次运行中均找到最优解；500 次迭代的单次运行消耗 CPU 时间 9 秒。

(6) 实例#6

问题[64]为：

$$\text{最小化} f(x) = 100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2+$$
$$10.1((x_2-1)^2+(x_4-1)^2)+19.8(x_2-1)(x_4-1)$$

服从约束：　$-10.0 \le x_i \le 10.0$，　$i=1,2,3,4$。

全局解为 $x^* = (1,1,1,1)$，$f(x^*) = 0$。GENOCOP 在所有的十次运行中均十分靠近最优解：找到的一个典型最优点为：$(x_1,x_2,x_3,x_4) = (1.000044, 1.000087, 0.999954, 0.999909)$，相应的目标函数值等于 0.00000001。10000 次迭代的单次运行消耗 CPU 时间 159 秒。

(7) 实例#7

问题[114]为：

$$\text{最小化}\quad f(x, \mathbf{y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5$$

服从约束：

$$x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \le 16$$
$$-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \le -1$$
$$2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 \le 24$$
$$0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 \le 12$$
$$-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \le 3$$
$$y_3 \le 1,\quad y_4 \le 1,\quad y_5 \le 2$$
$$x \ge 0,\quad y_i \ge 0,\quad (1 \le i \le 5)$$

全局最优解为$(x, y^*) = (0,6,0,1,1,0)$，$f(x, y^*) = -11$。GENOCOP 在所有的十次运行中均找到最优解；1000 次迭代的单次运行消耗 CPU 时间 23 秒。

(8) 实例#8

问题是由三个不相连的子问题[186]按照下面的方式构造而成：

最小化：

$$f(x) = \begin{cases} f_1 = x_2 + 10^{-5}(x_2 + x_1)^2 - 1.0, & 0 \le x_1 < 2 \\ f_2 = \dfrac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3 & 2 \le x_1 < 4 \\ f_3 = \dfrac{1}{3}(x_1 - 2)^3 + x_2 - \dfrac{11}{3} & 4 \le x_1 \le 6 \end{cases}$$

服从约束：

$$x_1/\sqrt{3} - x_2 \geq 0$$
$$-x_1 - \sqrt{3}x_2 + 6 \geq 0$$
$$0 \leq x_1 \leq 6 \quad 和 \quad x_2 \geq 0$$

函数 f 有三个全局解：$\boldsymbol{x}_1^* = (0,0)$、$\boldsymbol{x}_2^* = (3, \sqrt{3})$和$\boldsymbol{x}_3^* = (4,0)$。

在所有情况下，$f(x_i^*) = -1$（i=1, 2, 3）。我们作了三次分离的实算。在实算第 k（k=1, 2, 3）次中，除f_k以外的其他函数f_i增加0.5。结果，第一次实算的全局解为 $x_1^* = (0,0)$，第二次实算的全局解为 $x_2^* = (3, \sqrt{3})$，第三次实算的全局解为 $x_3^* = (4,0)$。GENOCOP 在所有运行中对这三种情况均找到全局最优解。500 次迭代的单次运行消耗 CPU 时间 9 秒。

小 结

如前所述，GENOCOP 对带线性约束的实算测试都工作得很好。但将 GENOCOP 推广到非线性约束处理上（即 *NP* 类问题），结果将如何还不很清楚。某些非线性约束仍然可以产生凸空间 — 该性质对许多算子（所有的变异、算术杂交）都是很重要的。但是，即使是在这种情况下，找到区间 *left*(*k*)和 *right*(*k*)的过程也是非常耗费计算时间的。另一种可能是用凸子空间族覆盖搜索空间（不必是不连贯的），并对它们中的每一个运行 GENOCOP。这种方法仍然有许多计算上的问题。

出于上述理由，我们更激励审视传统的优化方法。两个特殊的方法（在下一节中描述）将被选择与 GENOCOP 系统配合使用。

7.2 非线性优化：GENOCOP II

本节将讨论解决非线性规划问题的一个新的混合系统，GENOCOP II。所推出的系统在概念上是基于优化领域最新的发展[13]及 GENOCOP 的迭代执行（本节有时用 GENOCOP 代表 GENOCOP I 以区分于 GENOCOP II）。

基于微积分的方法假定目标函数 $f(\boldsymbol{x})$及约束都是 $\boldsymbol{x}$ 的二次连续可微函数。多数方法的一般做法是将非线性问题 *NLP* 转换为一系列可解子问题。包含在子问题的工作量在不同方法间变化较大。这些方法要求对显式（或者隐式）目标函数（或转换函数）进行二次导数计算，在某些方法里，目标函数可能是病态的，并引起算法失败。

在过去 30 年里，直接针对非线性优化问题已经有大量的研究，并在理论和实用中都取得一些进展[114]。在此领域，已经开发出来的一些方法包括：序贯二次罚函数法（sequential quadratic penalty function）[51,13]，递归二次规划法（recursive quadratic programming method）[40]，罚轨迹法（penalty trajectory methods）[291]和 SOLVER 法[112]。

这些方法中的一种，即序贯二次罚函数法是 GENOCOP II 系统的主要思想。此方法用问题 *NLP*′替换问题 *NLP* 如下：

$$\text{优化：} F(x, r) = f(x) + \frac{1}{2r}\bar{C}^T\bar{C}$$

这里 $r > 0$，$\bar{C}$ 为所有有效约束的向量 $c_1,...,c_l$。

Fiacco 和 McCormick[253]证明：*NLP* 和 *NLP'* 的解在 $r \to 0$ 的极限下是相同的，*NLP'* 能按 r 的降序正值的序列通过牛顿法求 $F(x, r)$的最小值来简化求解[111]。但这种方法是不稳定的，因为最小化 $F(x, r)$对较小的 r 值是极端无效的，Murray [291]证明这是由于当 $r \to 0$ 时，$F(x, r)$的 Hessian 矩阵将变得越来越病态。似乎没有显而易见的方法来解决这一问题，因此这个方法渐渐弃而不用。最近，Broyden 和 Attia 给出了一个方法和简单的二次罚函数结合在一起来克服数值困难[52,51]。搜索方向的计算不要求解线性方程组，这样，可以预计此方法要求的计算量要比其他一些算法少。该方法还提供了一个计算参数 r 的初始值及其后续值的自动技术[51]。

上述技术和 GENOCOP I 系统一起构成了一个新的系统 GENOCOP II。GENOCOP II 的结构如图 7.1 所示。

```
procedure GENOCOP II
begin
    t ←— 0
    split the set of constraints C into: C = L ∪ N_e ∪ N_i
    select a starting point x_s
    set the set of active constraints, A to A ←— N_e ∪ V
    set penalty  τ ←— τ_0
    while (not termination-condition) do
    begin
        t ←— t+1
        execute GENOCOP I for the function
              F(x,τ) = f(x) + (1/2τ) Ā^T Ā
                    with linear constraints L and the starting point x_s
        save the best individual x*: x_s ←— x*
        update A:  A ← A − S ∪ V,
        decrease penalty τ:  τ ← g(τ, t)
    end
end
```

图 7.1　GENOCOP II 的结构

算法执行的第一阶段（在它进入 **while** 循环前）分为几步：参数 t 被初始化为零（它记录算法迭代次数，即 GENOCOP I 应用的次数），约束集合 C 被分成三个子集：线性约束 L、非线性等式 N_e 及非线性不等式 N_i。随后是选择（或者用户输入）优化过程的开始点 x_s（不必是可行的）。约束集 A 初始是由 N_e 及 N_i 中违反约束的集合 $V \subseteq N_i$ 中的元素构成。在点 x，当且仅当 $c_j(x) > \delta$（其中 $j = p+1,..., m$），约束 $c_j \in N_i$ 是被违反的，这里δ为该方法的参数。最后，系统的初始罚系数τ被设定为 τ_0（方法中的一个参数）。

在算法的主要循环中，我们应用了 GENOCOP I 来优化带有线性约束 L 的修正函数：

$$F(x,\tau) = f(x) + \frac{1}{2\tau}\bar{A}^T\bar{A}$$

注意，GENOCOP I 的初始群体由第一次迭代的初始点及随后迭代保存的最好点的 *pop_size* 个同等拷贝组成；过程早期使用了几个变异算子以增加群体多样性。当

GENOCOP I 收敛，其最好个体 x^* 被保存并作为下一次迭代的开始点 x_s。但是，下一次迭代的执行伴随着惩罚参数值（$\tau \leftarrow g(\tau, t)$）的减小，及一套新的起作用的约束 A：

$$A \leftarrow A - S \cup V$$

其中 S 和 V 分别为 N_t 上被 x^* 满足和违反的子集。注意，τ 的减小将导致惩罚的增加。

算法的机制可由下面的例子来说明。问题为：

$$\text{最小化} f(x) = x_1 \cdot x_2^2$$

服从一个非线性约束

$$c_1: \ 2 - x_1^2 - x_2^2 \geq 0$$

已知全局最优解为 $x^* = (-0.816497, -1.154701)$ 及 $f(x^*) = -1.088662$。起始的可行点为 $x_0 = (-0.99, -0.99)$。经过 GENOCOP II 的第一次循环后（A 为空），系统收敛于 $x_1 = (-1.5, -1.5)$，$f(x_1) = -3.375$。因为点 x_1 违反了约束 c_1，于是此约束变成激活态。随后点 x_1 用来作为第二迭代的开始点。第二次迭代（$\tau = 10^{-1}$, $A = \{c_1\}$）产生点 $x_2 = (-0.831595, -1.179690)$，$f(x_2) = -1.122678$。然后点 x_2 被用来作为第三次迭代的开始点。第三次迭代（$\tau = 10^{-2}, A = \{c_1\}$）结果为 $x_3 = (-0.815862, -1.158801)$，$f(x_3) = -1.09985$。点 x_t（$t = 4, 5, \ldots$ 为算法的迭代次数）序列渐渐接近最优点。

为评价 GENOCOP II，我们选择了一套测试问题以验证算法的执行结果。并得出此算法在使用中是成功的。五个测试实例包括带若干非线性约束的二次、非线性及不连续函数。

系统所有的运行都是在 SUN SPARC station 2 上进行的。在所有实算中，对 GENOCOP I 使用下面的测试参数：

pop_size = 70（群体规模），k = 28（每代中亲体的数目），b = 6（非均匀变异系数），δ = 0.01（用以确定约束是否激活的参数）。

对多数实例，初始的罚系数 τ_0 设置为 1（即 $g(\tau, 0) = 1$）;且 $g(\tau, t) = 10^{-1} \cdot g(\tau, t-1)$。

对每个测试实例，GENOCOP II 都被执行十次。对多数问题，GENOCOP I 所需的收敛代数为 1000 代（较难的问题要求更大数量的迭代）。我们没有报告这些测试实例的计算时间，因为我们还没用完全完成 GENOCOP II。系统的行为是通过以手工形式执行其外部循环来模拟的：当 GENOCOP I 收敛，最好点作为下一次迭代的开始点，检查约束的激活状态；相应地调整评价函数。

(1) 测试实例#1

问题（取自[186]）为最小化 $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

服从非线性约束：

$$c_1: \ x_1 + x_2^2 \geq 0$$
$$c_2: \ x_1^2 + x_2 \geq 0$$

及边界约束：

$$-0.5 \leq x_1 \leq 0.5 \quad \text{和} \quad x_2 \leq 1.0$$

已知全局解为 $x^* = (0.5, 0.25)$，$f(x^*) = 0.25$。开始的可行点为 $x_0 = (0, 0)$，GENOCOP II 在一次迭代中即找到精确的最优。此时没有一个非线性约束被激活。

(2) 测试实例#2

问题（取自[113]）为最小化 $f(x, y) = -x-y$

服从非线性约束：

$$c_1:\ y \le 2x^4 - 8x^3 + 8x^2 + 2$$
$$c_2:\ y \le 4x^4 - 32x^3 + 88x^2 - 96x + 36$$

及边界约束：

$$0 \le x \le 3 \quad 和 \quad 0 \le y \le 4$$

已知全局解为 $\boldsymbol{x}^* = (2.3295, 3.1783)$，$f(\boldsymbol{x}^*) = -5.5079$。开始的可行点为 $\boldsymbol{x}_0 = (0, 0)$。可行区间几乎不连通。GENOCOP II 在第 4 次迭代非常接近最优解。系统的进展如表 7.1 所示。

表 7.1　GENOCOP II 在测试实例 #2 中的进展（对迭代 0 最好点为开始点）

迭代数	最好点	激活的约束
0	(0,0)	无
1	(3,4)	c_2
2	(2.06, 3.98)	c_1, c_2
3	(2.3298, 3.1839)	c_1, c_2
4	(2.3295, 3.1790)	c_1, c_2

(3) 测试实例#3

问题（取自[113]）为最小化 $f(\boldsymbol{x}) = (x_1-10)^3 + (x_2-20)^3$

服从非线性约束：

$$c_1:\ (x_1-5)^2 + (x_2-5)^2 - 100 \ge 0$$
$$c_2:\ -(x_1-6)^2 - (x_2-5)^2 + 82.81 \ge 0$$

及边界约束：

$$13 \le x_1 \le 100 \ 及 \ 0 \le x_2 \le 100$$

已知全局解为 $\boldsymbol{x}^* = (14.095, 0.84296)$，$f(\boldsymbol{x}^*) = -6961.81381$（见图 7.2）。开始点 $\boldsymbol{x}_0 = (20.1, 5.84)$是不可行点。

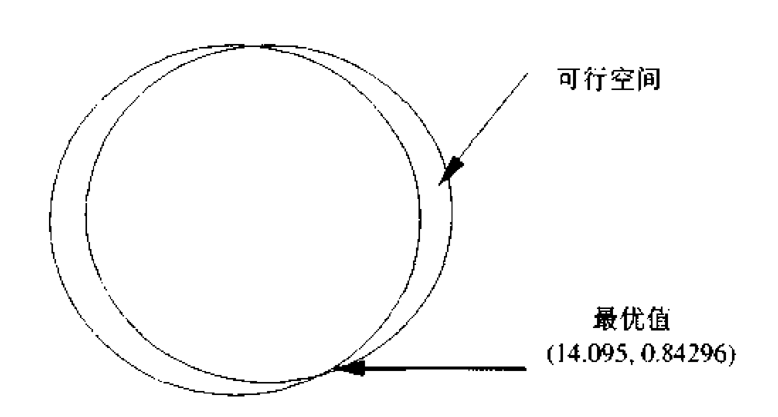


图 7.2　测试实例 #3 的可行空间

GENOCOP II 在第 12 次迭代时非常接近最优值。系统的进展如表 7.2 所示。

表 7.2 GENOCOP II 在测试实例#3 中的进展(对 0 迭代最好点即起始点)

迭代数	最好点	激活的约束
0	(20.1, 5.84)	c_1, c_2
1	(13.0, 0.0)	c_1, c_2
2	(13.63, 0.0)	c_1, c_2
3	(13.63, 0.0)	c_1, c_2
4	(13.73, 0.16)	c_1, c_2
5	(13.92, 0.50)	c_1, c_2
6	(14.05, 0.75)	c_1, c_2
7	(14.05, 0.76)	c_1, c_2
8	(14.05, 0.76)	c_1, c_2
9	(14.10, 0.87)	c_1, c_2
10	(14.10, 0.86)	c_1, c_2
11	(14.10, 0.85)	c_1, c_2
12	(14.098, 0.849)	c_1, c_2

(4)测试实例#4

问题(取自[39])为最小化 $f(x_1,x_2) = 0.01x_1^2 + x_2^2$

服从非线性约束:

$$c_1:\quad x_1x_2 - 25 \geq 0$$

$$c_2:\quad x_1^2 + x_2^2 - 25 \geq 0$$

及边界约束:

$$2 \leq x_1 \leq 50 \text{ 和 } 0 \leq x_2 \leq 50$$

全局解为 $x^* = (\sqrt{250}, \sqrt{2.5}) = (15.811388, 1.581139)$,$f(x^*) = 5.0$。开始点(不可行点)为 $x_0 = (2, 2)$。

有趣的是,标准的冷却方案 [即 $g(\tau, t) = 10^{-1} \cdot g(\tau, t-1)$] 并不能产生好的结果,但是,当冷却过程很慢时 [即 $g(\tau,0) = 5$ 和 $g(\tau, t) = 2^{-1} \cdot g(\tau, t-1)$],系统却很容易接近最优解(表 7.3)。当然,这会引发一些对给定问题有关如何控制温度的问题,这是以后要研究的一个课题。

表 7.3 GENOCOP II 在测试实例#4 中的进展(其中迭代 0 中最好点即起始点)

迭代数	最好点	激活的约束
0	(2, 2)	c_1, c_2
1	(3.884181, 3.854748)	c_1
2	(15.805878, 1.581057)	c_1
3	(15.811537, 1.580996)	c_1

(5)测试实例#5

最后的测试问题(取自[186])为最小化 $f(x) = (x_1-2)^2 + (x_2-1)^2$

服从一个非线性约束:

$$c_1:\quad -x_1^2 + x_2 \geq 0$$

及一个线性约束:

$$x_1 + x_2 \leq 2$$

全局解为 $x^* = (1,1)$,$f(x^*) = 1$。开始点(可行点)为 $x_0 = (0,0)$。GENOCOP II 在第 6 次迭代非常靠近最优点。系统的进展如表 7.4 所示。

表 7.4 GENOCOP II 在测试实例#5 中的进展（对迭代 0 最好点即起始点）

迭代数	最好点	激活的约束
0	(0, 0)	c_1
1	(1.496072, 0.503928)	c_1
2	(1.020873, 0.979127)	c_1
3	(1.013524, 0.986476)	c_1
4	(1.002243, 0.997757)	c_1
5	(1.000217, 0.999442)	c_1
6	(1.000029, 0.999971)	c_1

小 结

上述方法中有几个有趣的方面。首先，类似于其他任何基于遗传算法的方法，它不要求任何隐式（或者显式）的梯度计算或目标函数的 Hess 矩阵及约束计算。因此，这种方法不会遇到通常出现在基于微积分方法里的病态 Hess 矩阵的问题。

应该注意到，可以用任何遗传算法替换 GENOCOP II 中的内循环用到的 GENOCOP I。在这些测试实例中，所有的约束（线性或非线性）都应该考虑在激活的约束集 A 的设置（L 的成员应该分布在 N_e 和 N_i 之间）。但是这种方法慢得多而且不是很有效：出于效率的理由，最好单独处理线性约束（如 GENOCOP I 那样）。

7.3 其他技术

在过去两年里，有人建议了几种用遗传算法处理数值优化问题中的约束的方法。它们中的多数都是基于罚函数的概念，即惩罚不可行解①。

$$eval(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{若 } \boldsymbol{x} \text{ 是可行的} \\ f(\boldsymbol{x}) + penalty(\boldsymbol{x}) & \text{若 } \boldsymbol{x} \text{ 是不可行的} \end{cases}$$

如果没有违反约束的情况发生，$penalty(x)$为零，否则为正。对多数方法，函数集 f_j（$1\le j\le m$）被构造成罚函数；函数 f_j 按照下面方式度量第 j 个约束的违反程度：

$$f_j(\boldsymbol{x}) = \begin{cases} \max\{0, g_j(\boldsymbol{x})\} & \text{若 } 1\le j\le q \\ |h_j(\boldsymbol{x})| & \text{若 } q+1\le j\le m \end{cases}$$

但是，至于罚函数如何设计及如何应用于不可行解，这些方法在许多重要细节上有差别。在下面部分，我们将依次讨论它们；这些方法按照它们所需参数的降序排列。

(1) 方法#1

此方法是由 Homaifar 等[195]建议。方法假定对每个约束，可以建立一族间隔以确定适当的惩罚系数。其步骤如下：

- 对每个约束，产生若干（l）水平的违约；
- 对每个违约水平及每个约束，产生一个惩罚系数 R_{ij}（$i = 1,2,\dots, l; j = 1,2,\dots,m$）；较高水平的违约产生较大的系数值。

① 在本节的其余部分，我们假定为求最小问题。

- 以随机的个体群体（可行的或不可行的）为起点；
- 演化群体：用下面公式评价个体

$$eval(x) = f(x) + \sum_{j=1}^{m} R_{ij} f_j^2(x)$$

方法的缺点是参数的数量：对 m 个约束，此方法要求 m 个参数作为对每个约束建立的区间数（在[195]中，对所有 $l = 4$ 的约束，这些参数是相同的），对每个约束需要 l 个参数，（即总共有 $l \times m$ 个参数；这些参数表示区间的边界或违约水平），再加上每个约束还需要 l 个参数（这些参数代表惩罚系数 R_{ij}）。所以，方法总共要求 $m(2l + 1)$ 个参数来处理 m 个约束。对 m =5 个约束、l = 4 个违约水平，需要 45 个参数！很明显，结果与参数密切相关。很有可能对一给定问题，存在使系统返回最靠近可行的近似最优解的参数集，但是很可能难以找到它。

(2) 方法#2

第二个方法是由 Joines 和 Houck [210]建议的，和前面的方法相对，作者假设的是动态惩罚。个体按照下面的公式被评价（在第 t 迭代）：

$$eval(x) = f(x) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(x)$$

其中 C，α和β为常数。对这些参数的一个合理的选择是（在[210]中有报道）$C = 0.5$，$\alpha = \beta = 2$。此方法比第一种方法要求少得多的参数（与约束数目无关）。另外，它不定义违约水平，不可行解的压力随惩罚项分量$(C \times t)^{\alpha}$增加：在过程的结尾（较高的代数值 t），此分量呈现较大值。

(3) 方法#3

第三个方法是由 Schoenauer 和 Xanthakis [346]建议的：其工作步骤如下：

- 从一个个体组成的随机群体开始（可行的或不可行的）；
- 设置 j =1（j 为约束计数器），
- 用 $eval(x) = f_j(x)$演化该群体，直到对此约束该群体以给定的百分数（所谓的触发阀值 $\phi - flip\ threshould$）是可行的①；
- 设置 $j = j + 1$；
- 当前群体为下一阶段演化的开始点，这里 $eval(x) = f_j(x)$。在此阶段，不满足第 1、第 2、...或第 j–1 约束中的一个的点从群体中消去。终止判据是群体以触发阀值百分数 ϕ 再次满足第 j 个约束；
- 如果 $j < m$，重复最后两步，否则（$j = m$）优化目标函数，即 $eval(x) = f(x)$，排除不可行个体。

此方法要求所有约束按线性次序依次被处理。不清楚的是约束的次序如何影响算法的执行结果；我们的实算表明不同的次序得到不同的结果（在总运行时间和精度的意义上有差别）。

① 此方法建议使用共享模式（以维持群体多样性）。

总的来说，此方法要求 3 个参数：共享因子σ，触发阈值ϕ及特定的约束次序。这种方法和前面两种方法十分不同，总体上也与其他惩罚方法不同，因为它一次只考虑一个约束。另外，在算法的最后阶段，该方法优化目标函数 f 本身，而不带有任何惩罚因素。

(4) 方法#4

我们把 GENOCOP Ⅱ系统定为方法#4。如前所述，这是本书描述的仅有的一种区分线性和非线性约束的方法。这种算法用一套闭算子维持所有线性约束的可行解，它将可行解（只是对线性约束可行解）转变为另一种可行解。算法的每次迭代只考虑起作用的约束，不可行解的压力随温度值τ的增加而增加。

该方法有一个附加特性：它开始于一单个点①。因此，把这种方法和其他由某开始点进行试验的经典优化方法做比较是相对容易的（对给定问题）。

这种方法要求开始的和"冷冻"的温度分别为τ_0 和τ_f，以及降低温度τ的冷却方案。标准值（在[267]中）为：对$\tau_f = 0.000001$，$\tau_0 = 1, \tau_{i+1} = 0.1 \cdot \tau_i$。

(5) 方法#5

第五种方法是由 Powell 和 Skolnick[313]开发的。这种方法为一带有显著的例外色彩的经典的惩罚方法。每个个体按照下面公式被评价：

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) + r \sum_{j=1}^{m} f_j(\boldsymbol{x}) + \lambda(t, \boldsymbol{x})$$

其中 r 为一常数；还要另一分项$\lambda(t, \boldsymbol{x})$为一附加的有关函数，它影响对不可行解的评价。其要点是采用一附加的启发规则（较早在[332]中提出）：对任意可行个体 $\boldsymbol{x}$ 和任意不可行个体 $\boldsymbol{y}$，$eval(\boldsymbol{x}) < eval(\boldsymbol{y})$，即任何可行解比任何不可行解好。这可通过许多方式完成；一种可能的方式是设置：

$$\lambda(t, \boldsymbol{x}) = \begin{cases} 0 & 若\ \boldsymbol{x} \in F \\ \max\{0, \max_{x \in F}\{f(\boldsymbol{x})\} - \min_{x \notin F}\{f(\boldsymbol{x}) + r\sum_{j=1}^{m} f_j(\boldsymbol{x})\}\} & 若\ \boldsymbol{x} \notin F \end{cases}$$

其中 F 表示搜索空间可行部分。换句话说，对不可行个体惩罚增加：使它们的评价值不好于可行个体最差值（即 $\max_{x \in F}\{f(\boldsymbol{x})\}$）②。

(6) 方法#6

最后这种方法排除不可行解（死亡惩罚）；这种方法也被进化策略[18]、用于数值优化的演化规划[117]及模拟退火采用。

7.3.1 五个测试实例

在选择下面五个测试实例过程中，我们考虑了：（1）目标函数的类型；（2）变量

---

① 然而，这种性质不是基本的。唯一重要的要求是：下一群体包含前面群体中的最好个体。

② Powell 和 Skolnick 通过把可行解的评价值映射到区间 $(-\infty, 1)$里；把不可行解的评价值映射到区间 $(1, \infty)$里，可达到同样的结果。对加权选择和竞争选择，这种实现方法上的不同是不重要的。

的数目；（3）约束的数目；（4）约束的类型；（5）在优化激活的约束数目；（6）可行搜索空间占整个搜索空间的比率 ρ。我们并不想声明这些已提出的测试实例 G1~G5 是完整的，但是它们构成了一套能灵活对其他约束处理方法进行基本测试的测试集。

(1) 测试实例#1

问题[114]是求函数的最小：

$$G1(x)=5x_1+5x_2+5x_3+5x_4-5\sum_{i=1}^{4}x_i^2-\sum_{i=5}^{13}x_i$$

服从于约束：

$$2x_1+2x_2+x_{10}+x_{11}\le 10,\qquad 2x_1+2x_3+x_{10}+x_{12}\le 10$$
$$2x_2+2x_3+x_{11}+x_{12}\le 10,\qquad -8x_1+x_{10}\le 0$$
$$-8x_2+x_{11}\le 0,\qquad -8x_3+x_{12}\le 0$$
$$-2x_4-x_5+x_{10}\le 0,\qquad -2x_6-x_7+x_{11}\le 0$$
$$-2x_8-x_9+x_{12}\le 0,\qquad 0\le x_i\le 1,\quad i=1,\dots,9$$
$$0\le x_i\le 100,\quad i=10,11,12,\qquad 0\le x_{13}\le 1$$

问题中有 9 个线性约束；函数 G1 为二次函数，其全局最小为：

$$x^*=(1,1,1,1,1,1,1,1,1,3,3,3,1)$$

此时 $G1(x^*)=-15$。9 个约束中的 6 个在全局最优点处起作用（激活），未起作用的是下面 3 个约束：$-8x_1+x_{10}\le 0$，$-8x_2+x_{11}\le 0$，$-8x_3+x_{12}\le 0$。

(2) 测试实例#2

问题[186]是求函数最小：

$$G2(x)=x_1+x_2+x_3$$

服从于

$$1-0.0025(x_4+x_6)\ge 0,\qquad 1-0.0025(x_5+x_7-x_4)\ge 0$$
$$1-0.01(x_8-x_5)\ge 0,\qquad x_1x_6-833.33252x_4-100x_1+83333.333\ge 0$$
$$x_2x_7-1250x_5-x_2x_4+1250x_4\ge 0,\quad x_3x_8-1250000-x_3x_5+2500x_5\ge 0$$
$$100\le x_1\le 10000,\qquad 1000\le x_i\le 10000,\ i=2,3$$
$$10\le x_i\le 1000,\quad i=4,\dots,8$$

问题中有 3 个线性和 3 个非线性约束；函数 G2 为线性，其全局最小为：

$$x^*=(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$$

此时 $G2(x^*)=7049.330923$。所有的 6 个约束在全局最优处都起作用。

(3) 测试实例#3

问题[186]是求函数最小：

$$G3(x)=(x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7,$$

服从于：

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0,$$
$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0,$$
$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0,$$
$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$
$$-10.0 \leq x_i \leq 10.0, \quad i = 1, \ldots, 7。$$

问题中有 4 个非线性约束：函数 G3 为非线性，其全局最小为：

$$x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$$

此时 $G3(x^*) = 680.6300573$。4 个约束中的 2 个（第 1 和第 4）在全局最优处起作用。

(4) 测试实例#4

问题[186]是求函数最小：

$$G4(x) = e^{x_1x_2x_3x_4x_5}$$

服从于：

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10, \qquad x_2x_3 - 5x_4x_5 = 0$$
$$x_1^3 + x_2^3 = -1, \qquad -2.3 \leq x_i \leq 2.3, \ i = 1,2$$
$$-3.2 \leq x_i \leq 3.2, \quad i = 3,4,5$$

问题中有 3 个非线性等式；非线性函数 G4 全局最小点位于

$$x^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.7636450)$$

此时 $G4(x^*) = 0.0539498478$。

(5) 测试实例#5

问题[186]为求函数最小：

$$G5(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2$$
$$+ 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

服从于

$$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0$$
$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$
$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$
$$-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$
$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$$
$$-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$
$$-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$
$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$$
$$-10.0 \leq x_i \leq 10.0, \quad i = 1, \ldots, 10$$

问题中有 3 个线性和 5 个非线性约束：G5 为二次函数，其全局最优点位于

$$x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548,$$
$$1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$$

此时 $G5(x^*) = 24.3062091$。8 个约束中的 6 个（除最后 2 个）在全局最优处起作用。

小 结

所有的测试实例概述在表 7.5 中；对每个测试例子（TC），我们列出了变量数 n、函数 f 的类型、可行搜索空间占整个搜索空间大小的比率 ρ、每类约束的数目（线性不等式数 LI、非线性等式数 NE 及不等式数 NI）及在最优点起作用的约束数目 a。

表 7.5 五个测试实例小结

TC	n	函数 f 类型	ρ	LI	NE	NI	a
#1	13	二次型	0.0111%	9	0	0	6
#2	8	线　性	0.0010%	3	0	3	6
#3	7	多项式	0.5121%	0	0	4	2
#4	5	非线性	0.0000%	0	3	0	3
#5	10	二次型	0.0003%	3	0	5	6

注：比率 ρ 通过取搜索空间中产生 1 000 000 个随机点并检验它们是否可行，从而被实验性地确定；LI，NE 和 NI 分别表示线性不等式数、非线性等式数和不等式数。

7.3.2 实 算

在所有的实算中，我们采用浮点表达、非线性分级加权选择、Gauss 变异、算术及启发式杂交；所有算子的概率都被设定为 0.08，群体规模为 70。对所有的方法，系统运行 5000 代。

表 7.6 实算结果

TC	精确最优值		方法 #1	方法 #2	方法 #3	方法 #4	方法 #5	方法 #6	方法 #6 (f)
#1	−15.000	b	−15.002	−15.000	−15.000	−15.000	−15.000		−15.000
		m	−15.002	−15.000	−15.000	−15.000	−15.000	—	−14.999
		w	−15.001	−14.999	−14.998	−15.000	−14.999		−13.616
		c	0,0,4	0,0,0	0,0,0	0,0,0	0,0,0		0,0,0
#2	7049.331	b	2282.723	3117.242	7485.667	7377.976	2101.367		7872.948
		m	2449.798	4213.497	8271.292	8206.151	2101.411	—	8559.423
		w	2756.679	6056.211	8752.412	9652.901	2101.551		8668.648
		c	0,3,0	0,3,0	0,0,0	0,0,0	1,2,0		0,0,0
#3	680.630	b	680.771	680.787	680.836	680.642	680.805	680.934	680.847
		m	681.262	681.111	681.175	680.718	682.682	681.771	681.826
		w	689.660	682.798	685.640	680.955	685.738	689.442	689.417
		c	0,0,1	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
#4	0.054	b	0.084	0.059		0.054	0.067		
		m	0.955	0.812	*	0.064	0.091	*	*
		w	1.000	2.542		0.557	0.512		
		c	0,0,0	0,0,0		0,0,0	0,0,0		
#5	24.306	b	24.690	25.486		18.917	17.388		25.653
		m	29.258	26.905	—	24.418	22.932	—	27.116
		w	36.060	42.358		44.302	48.866		32.477
		c	0,1,1	0,0,0		0,1,0	1,0,0		0,0,0

注：对每个方法（#1、#2、#3、#4、#5 和#6），我们报告了最好(b)、中等(m)及最差(w)结果（十次独立的运行）和在中等解时出现的违约数(c)：三个数字序列分别表示大于 1.0、大于 0.1 和大于 0.001 的违约数。符号“*”和“—”分别表示“方法未用在此测试实例”和“结果毫无意义”。对方法#6 还进行了可行初始群体实算(译者注：原书中此表是由两个表 Fig7.6 和 Fig7.7 组成）。

结果在表 7.6 中进行了总结，它报告了对每种方法最好（行 b）、中等（行 m）及最差（行 w）结果（十次独立的运行）及在中间解中违约的次数（行 c）：3 个数字的序

列分别表示在 1.0 和 10.0 之间、0.1 和 1.0 之间以及 0.001 和 0.1 之间违约数（3 个零的序列表示一个可行解）。如果至少一个违约是大于 10.0（对函数 f_j）的，解就被考虑成“毫无意义”。在某些情况下，由于目标值和违约数之间的关系，很难确定“最好解”：表中报告了最小目标值（对最好解）；因此，在全局最小值处，可能一些值比该值“更好”。

很难对基于 5 个测试实例的所有 6 种方法提供完整的分析。只有当违约水平和惩罚系数 R_{ij} 同问题协调得很好时，似乎方法#1 可以提供好的结果（我们任意对惩罚分别选择 100，200，500，1000 四个违约水平，对测试实例 #1 和#3，算法工作得很好，而对其他测试例子不好，此时一些其他参数值需要改变）。同样，这些违约水平和罚系数不能阻止系统在测试实例#2 中违反 3 个约束，因为“奖励”太大而不能有效地惩罚（即用相对较小的惩罚向量 $\boldsymbol{x}$ = (100.00,1000.00,1000.00,128.33,447.95,336.07,527.85, 578.08)，得到可行范围以外的最优值为 2100）。在除测试#4 以外的所有测试实例，对相对较小的边界,方法#1 返回的是不可行解，这是这种方法的一个有趣特征。

对几乎所有测试实例，方法#2 比方法#1 给出更好的结果：对测试实例#1（所有返回的解都是可行的）、测试实例#2 和#4（违约度较小）及测试实例#3（结果的标准偏差小得多）。另一方面，方法#2 的惩罚似乎太强了：因子$(C\times t)^{\alpha}$生长的太快而用处不大，系统几乎没有机会从局部最优中跳出：在多数实算中，最好个体在较早的代中被找到。值得一提的是这种方法对目标函数是二次型的测试例子#1 和#5 给出很好的结果。

方法#3 不能应用于测试实例#4，对测试实例#5 也不能给出有意义的解，其中比率ρ（区别于实例#4，ρ是最小的）非常小。很明显，此方法对搜索空间可行部分的尺寸十分敏感。同样，某些附加实算表明约束的次序在很大程度上影响最终的结果。另外，这种方法对测试实例#1 和#3 执行得很好，并对测试实例#2 给出合理的结果。

方法#4 对所有的测试实例#1、#3 和#4 都执行的很好，提供的解也是最好的。它同样给出测试实例#2 合理的结果（其中线性约束导致方法#1 和#2 失败）。但是，对测试实例#5，该方法比起方法#1，#2 和#6 来，却给出较差的结果：似乎，该问题的线性约束阻止系统向最优解靠近。这是一个有趣的例子，对群体只在可行（对应于线性约束）区域内进行的限制起破坏作用。附加的实算表明这种方法对冷却方案十分敏感。例如，测试实例#5 用不同的冷却方案（$\tau_{i+1}=0.01\cdot\tau_i$）得到的结果有很大程度的改进。

方法#5 对测试实例#2 难以找到可行解：和方法#1、方法#2 相似，此算法总停留于不可行解。在所有其他测试实例中，这种方法给出一稳定的、合理的执行结果、返回可行解（测试实例#1、#3 和#5）或者轻度的不可行解（测试实例#4）。其他实算（在表中未列出）包括用可行初始群体运行方法#5。对测试实例#2，结果几乎和方法#6 获得的结果相同。但对测试实例#5，其结果在所有方法中是最杰出的。

除测试实例#3 以外，方法#6 不能产生任何有意义的结果。为适当地测试这种方法，有必要用可行解来初始化群体（方法#6）。不同的初始化方案使比较这些方法更为困难：不过有一种模式显露出来：这些方法一般都给出十分差的执行。这种方法不像其他方法，它对最容易的测试实例#1 是不稳定的（它是唯一返回解-13.616 的方法，该解远离最优解）。对实例#2，只有一次运行返回的值小于 8000。

没有一个单参数（线性、非线性、起作用的约束个数、比率ρ、函数类型、变量数）被证明在演化技术中对度量问题的难度有特殊的意义。例如，所有的方法对测试实例#1和#5（其中 $\rho = 0.0111\%$ 和 $\rho = 0.0003\%$）都十分靠近最优解，而多数方法对测试实例#2（其中$\rho = 0.0010\%$）实算都难以靠近最优解。带有相似约束数目（分别为 9 个和 8 个）和相同数目的最优点起作用约束（6 个）的两个二次优化函数（测试实例#1 和#5），对多数方法都提出程度不同的挑战。看起来是其他性质（目标函数的特征及可行区域的拓扑性质）构成了问题难度的十分重要的度量。另外，其中的几个方法对初始群体中的可行解的提供十分敏感。

7.4 其他可能性

如前所述，几位研究者研究了罚函数设计中的经验。一些假设在[332]中进行了明确的表达：

- "作为与可行性距离函数有关的惩罚要比只与违约数函数有关的惩罚好；
- 对只有很少的约束及只有很少完全解的问题，只与违约数函数有关的惩罚不大可能找到解；
- 构筑好的罚函数应该考虑两种性质：最大完成开销及期望完成开销；
- 惩罚应该接近期望完成开销，但不应该经常低于它；惩罚越精确，找到的解越好。当惩罚总是低估完全开销时，搜索可能找不到一个解。"

另外，在[358]中叙述道：

- "带有可变惩罚系数的遗传算法要比固定惩罚因子的算法执行得好。"

其中，罚系数的可变性是由经验性规则确定的。

Smith 和 Tate[360]较深入地研究了最后这一评述。在他们的工作中用动态惩罚实算，其中的惩罚根据违约数变化，找到了最好的可行目标函数及最好的目标函数值。

还有，Bean 和 Hadj-Alouane[27,174]开发出一个自适应惩罚的方法。此方法使用了一个罚函数，罚函数的一个分量由搜索过程的反馈信息得出。每个个体由下面的公式进行评价：

$$eval(\bar{x}) = f(\bar{X}) + \lambda(t)\sum_{j=1}^{m} f_j^2(\bar{x})$$

其中，$\lambda(t)$按照下面的方法对每代 t 进行更新：

$$\lambda(t+1) = \begin{cases} \frac{1}{\beta_1}\cdot\lambda(t) & \text{若对所有的 } t-k+1 \le i \le t,\ \bar{B}(i) \in F \\ \beta_2\cdot\lambda(t) & \text{若对所有的 } t-k+1 \le i \le t,\ \bar{B}(i) \notin F \\ \lambda(t) & \text{其他} \end{cases}$$

其中，$\vec{B}(i)$表示函数 *eval* 在代 i，$\beta_1, \beta_2 > 1$ 及$\beta_1 \ne \beta_2$（避免循环）的最好个体。换句话说，该方法为：（1）如果在最后 k 代里所有最好个体都是可行的，则减小对 t+1 代的惩罚$\lambda(t+1)$；（2）如果在最后 k 代里的所有最好个体都是不可行的，则增加其惩罚。如果

一些可行个体和不可行个体在最后 k 代里是最好个体，$\lambda(t+1)$保持不变。

上述方法已应用于整数规划问题，在[174]考虑的问题是，

$$\text{最小化：} \boldsymbol{cx}$$

服从约束：

$$\boldsymbol{Ax}-\boldsymbol{b}\geq 0 \quad (1)$$

$$\sum_{j=1}^{n_i} x_{ij}=1,\quad \text{其中 } i=1,\ldots,m \quad (2)$$

$$x_{ij}\in\{0,1\} \quad (3)$$

对每一设定（$i=1,\ldots,m$），等式（2）和（3）精确地强制$\{x_{ij}\}_{j=1}^{n_i}$中的一个变量为 1。矩阵 A 为 $k\times n$ （$n=\sum_{i=1}^{m} n_i$，$i=1,\ldots,n$），$\boldsymbol{b}$ 为一固定的 k 维向量。

[174]中报道了解决上述整数规划问题比较成功的方法是分支定界法 (branch and bound methods)，它们用到线性规划、Lagrange 松弛法或它们的变种方法之一。Lagrange 松弛法通过加权线性惩罚合并违反约束来消去一些约束。“正确”的权值能导致非常好的边界甚至是原始问题的最优解；一个典型的 Lagrange 松弛法按照下面公式替换原始问题［约束（1）］

$$\text{求最小：} \boldsymbol{cx}-\boldsymbol{\lambda}(\boldsymbol{Ax}-\boldsymbol{b})$$

服从约束

$$\boldsymbol{Ex}=\boldsymbol{e}_m,\quad x_{ij}\in\{0,1\}$$

（约束 $\boldsymbol{Ex}=\boldsymbol{e}_m$ 为多重选择约束，其中 $\boldsymbol{e}_m$ 是值为 1 向量）。

所建议的方法这样替换原始问题

$$\text{最小化：} \boldsymbol{cx}+p_\lambda(\boldsymbol{x})$$

服从约束：

$$\boldsymbol{Ex}=\boldsymbol{e}_m,\quad x_{ij}\in\{0,1\}$$

其中 $p_\lambda(\boldsymbol{x})=\sum_{i=1}^{k}\lambda_i\,[\min\{0,A_i\boldsymbol{x}-b_i\}]^2$。函数是非线性的，而且利用遗传算法来优化表达式。

在上述建议的方法中含有一些有趣的思想。首先，实算结果表明，以较高值的$\boldsymbol{\lambda}$开始不能产生有效的算法。所以，建议的算法在运行过程中调整向量$\boldsymbol{\lambda}$：使用了递增的向量$\boldsymbol{\lambda}$序列。实算[174]表明，该序列增加的比率是很重要的（相似的观察可以从基于 GENOCOP II 中的实算中获得）：较慢的比率可以改进解的质量，但改进比率的消耗大；一个较快的比率可能导致无效的遗传演化。另外，建议的遗传算法使用了随机钥匙技术，其中一个解被表达成随机数的向量：它们以排列次序对解进行解码（注意这种方法和演化策略应用于货郎担问题的相似性；见第 8 章有关演化策略的讨论和第 10 章有关货郎担问题的讨论）。一个解被长度等于多重选择设定数的一个串表达。串中的每个位置 i 可以是$\{1,\ldots,n_i\}$中的任何整数，其中 n_i 为多选择设定的变量数。

其他约束处理方法也值得注意。其中之一（GENOCOP III，将在下一节讨论）是基于修补算法：一个不可行解 $\boldsymbol{x}$ 被“强制”进入可行范围并对其修补版本进行评价。同样可以构建一种合并某些确定性优化过程的混和算法；有关这类特殊方法的概观和描述，见[294]。

另一种可能是包含使用目标函数 f 及惩罚 f_j 的值作为一向量的元素，并应用多目标

技术求此向量所有元素的最小值。换句话说，目标函数 f 及违约度量 f_j（对 m 个约束）构成了一个（m+1）维向量 $\boldsymbol{v}$：

$$\boldsymbol{v}=(f, f_1, \ldots, f_m)$$

使用多目标优化方法，我们可以求其成员的最小值：对 $1 \leq i \leq m$ 及 $f(x) \leq f(y)$，对所有可行的 y（求最小问题）的一个理想解 x 将有 $f_j(x)=0$。这种方法的一个成功的实现最近由 Surry 等提出[376]。

另一种方法最近由 Le Riche 等提出[239]。作者设计了一种（分离）遗传算法，其中对每个约束使用了两个惩罚参数值，而不是一个；这两个值主要是通过维持个体的两个子群体来完成严厉惩罚和适度惩罚之间的平衡。群体被分割成两个协作组，其中每组中的个体使用两个惩罚参数之一进行评价。

最近，Paredis[308]报告了一种有趣的方法。该方法（在下文约束满足问题中有描述）是基于协作演化模型，其中一个潜在解群体和约束群体协作演化：更适的解满足更多约束，反之更适的约束被更多的解违反。它的意思是，解群体中的个体从整个搜索空间的角度被考虑，在可行个体和不可行个体之间没有区别。一个个体的评价是基于违约度量 f_j 来确定的；但是，较好的 f_j（即激活态约束）将更有利于解的评价。用这种方法处理数值优化问题并与其他方法比较将是有趣的。但采用这种方法要解决的一个主要困难和许多其他方法是非常一致的：如何平衡解的可行性压力和目标函数最小化压力之间的矛盾。

促使人们对文化算法(cultural algorithms)[327, 328, 330, 331]进行研究是基于这样的观察：文化可能是另一种类型的继承系统。但不十分清楚的是：表达文化信息适应和传送的最适当的结构和单元是什么。同样不清楚的是如何描述自然进化和文化之间的相互作用。Reynolds 开发了几个模型来研究文化算法的性质；在这些模型里，信念空间(belief space)被用来约束个体可以承担的特征组合。信念空间的改变表达宏观进化的变化，个体群体的变化表达微观进化变化。两种变化通过通讯链被稳固地节制。

信念空间背后的总的直觉是和特征水平上的“可接受”行为一起保存这些信念（因此，可砍去不可接受信念）。可接受信念作为支配特征群体的约束。看起来，文化算法可以当作解决数值优化问题的工具，这里约束以直接的方式影响搜索（因此在约束空间里的搜索可能比在不约束空间里的搜索更有效！）。最近 Reynolds 等[329]研究了应用文化算法处理约束数值优化问题的可能性。第一个计算实验表明这种方法有很大的潜力。

7.5 GENOCOP III

该方法合并了原始的 GENOCOP 系统（在 7.1 节中进行了描述），同时还通过维持两个分离的群体对原有的 GENOCOP 系统进行了扩展，其中一个群体的进展影响对另一个群体中个体的评价。第一个群体 P_s 由所谓的满足线性的搜索点组成（如原始的 GENOCOP 系统）。这些点的可行性（在线性约束的意义上）像以前那样通过特殊的算子来维持。第二个群体 P_r 是由所谓的参照点组成；这些点都是完全可行的，即它们满足

所有的约束（如果 GENOCOP III 开始时难以找到参照点，用户必须进行提示。通常在这种情况下，可行解和整个搜索空间大小的比率ρ非常小，可能发生的是参照点的初始设定是由单个可行点的多次拷贝组成）。图 7.3 说明了这两个群体。

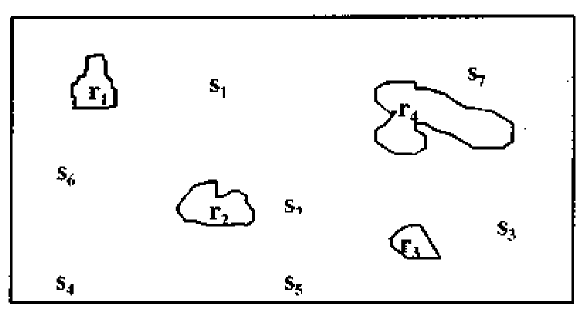


图 7.3　群体 $P_S=\{\vec{S}_1,\vec{S}_2,\vec{S}_3,\vec{S}_4,\vec{S}_5\}$ 和群体 $P_r=\{\vec{R}_1,\vec{R}_2,\vec{R}_3,\vec{R}_4\}$

参照点 $\vec{R}$ 是可行点，通过目标函数来直接评价（即 $eval(\vec{R})=f(\vec{R})$）。而为了评价不可行的搜索点，必须进行"修补"。修补过程如下。假定，有个搜索点 $\vec{S}$ 是不完全可行的。在这种情况下，系统选择一个参照点（较好的参照点有更多的机会被选择；一种选择的方法是基于非线性加权的），譬如 $\vec{R}$，从 $\vec{S}$ 和 $\vec{R}$ 之间的片断中通过产生在区间 ⟨0,1⟩ 里的随机数 a 的方法随机产生一点 $\vec{Z}$：$\vec{Z}=a\vec{S}+(1-a)\vec{R}$。图 7.4 说明了这一修补过程。

一旦找到一个可行解 $\vec{Z}$，$eval(\vec{S})=eval(\vec{Z})=f(\vec{Z})$[①]。另外，如果 $f(\vec{Z})$ 好于 $f(\vec{R})$，那么点 $\vec{Z}$ 替换 $\vec{R}$ 作为一个新的参照点。还有，$\vec{Z}$ 以某个概率 p_r 替换 $\vec{S}$。

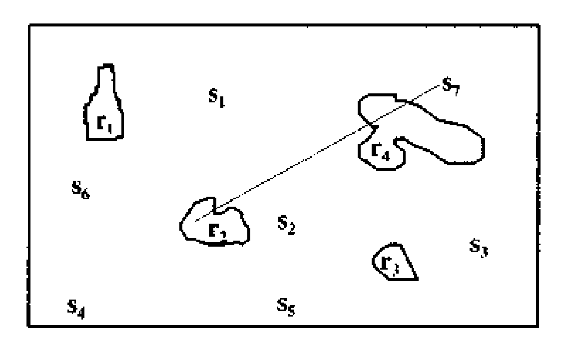


图 7.4　对不可行点 $\vec{S}$ 的评价

GENOCOP III 避免了其他系统的缺点。只引入了较少的附加参数（参照点的群体规模，替换概率），且总是返回可行解。搜索点通过参照，搜索在可行的搜索空间里进行。较好参照点的附近空间被更经常地开发。一些参照点被移入搜索点群体，其中要用一些特殊的算子（保持线性约束）对它们进行变换。

GENOCOP III 的初级版（可从网址 ftp.uncc.edu 目录 coe/evol 下的文件 genocopIII.tar.Z 中获得）经测试实例（在 7.3.1 中给出）G1~G5 测试。GENOCOP III 对 G1 的结果等同于原始的 GENOCOP 系统的结果：因为不存在非线性约束，不需要参照点群体。对其余的四个测试实例，我们实算了三个（G2, G3 和 G5）；问题 G4 包含非线性等式 *NE*，当前版本的 GENOCOP III 还不能处理它们。

① 很明显，由于修补过程的随机性的缘故，不同的代对同样的搜索点 $\vec{S}$ 可能会得到不同的评价值。

GENOCOP III 是在 5000 次迭代下运行的（类似于 7.3 中讨论的其他系统）。所有的算子概率被设定为 0.08，所有的群体规模为 70。

结果非常好。如对问题 G2，最好结果为 7286.650，比 7.3 节讨论的最好系统的最好结果（GENOCOP II 对该问题的解为 7377.976）还好得多。对其他两个问题 G3 （解 680.640）和 G5 （解 25.883）结果也是相似的。另外一个有趣的结果是与系统的稳定性有关。GENOCOP III 答案的标准偏差非常小。例如对问题 G3，所有的结果都在 680.640 和 680.889 之间；而其他系统产生各种各样的结果（在 680.642 和 689.660 之间，见[266]）。

当然，所有的结果点 $\vec{X}$ 都是可行的，而对其他系统，却不存在这种情况（例如 GENOCOP II 对问题 G2 产生的值为 18.917，而基于 Homaifar，Lai 和 Qi，Powell 和 Skolnick 的系统对问题 G2 得到的结果分别为 2282.723 和 2101.367）。

最近还出现一个有趣的测试实例：问题[220]是求一个函数的最大：

$$f(\boldsymbol{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

其中 $\prod_{i=1}^{n} x_i > 0.75$，$\sum_{i=1}^{n} x_i < 7.5n$，和 $0 < x_i < 10$，$1 \le i \le n$。

问题有两个非线性约束：函数 f 是非线性的，其全局最大是未知的。

Keane[220]提到：

“我当前正在使用一种 12 位二进制编码的并行遗传算法，并使用了杂交、倒位、变异、好位成形（niche forming）及修正的 Fiacco-McCormick 约束罚函数来处理它。对 n=20，经过 20000 代进化，我得到 0.76 这样的值。”

GENOCOP III 对该实例分别用 n= 20 和 n=50 运行。 对前者，10,000 代内找到的最好解为：

$\boldsymbol{x}$ = (3.16311359, 3.13150430, 3.09515858, 3.06016588, 3.03103566,
2.99158549, 2.95802593, 2.92285895, 0.48684388, 0.47732279,
0.48044473, 0.48790911, 0.48450437, 0.44807032, 0.46877760,
0.45648506, 0.44762608, 0.44913986, 0.44390863, 0.45149332),

此时 $f(\boldsymbol{x})$ = 0.80351067。对后者（n = 50），10000 代之内找到在最好解为：

$\boldsymbol{x}$ = (6.28006029, 3.16155291, 3.15453815, 3.14085174, 3.12882447,
3.11211085, 3.10170507, 3.08703685, 3.07571769, 3.06122732,
3.05010581, 3.03667951, 3.02333045, 3.00721049, 2.99492717,
2.97988462, 2.96637058, 2.95589066, 2.94427204, 2.92796040,
0.40970641, 2.90670991, 0.46131119, 0.48193336, 0.46776962,
0.43887550, 0.45181099, 0.44652876, 0.43348753, 0.44577143,
0.42379948, 0.45858049, 0.42931050, 0.42928645, 0.42943302,
0.43294361, 0.42663351, 0.43437257, 0.42542559, 0.41594154,
0.43248957, 0.39134723, 0.42628688, 0.42774364, 0.41886297,
0.42107263, 0.41215360, 0.41809589, 0.41626775, 0.42316407),

其中 $f(x) = 0.83319378$ ①。

很明显，GENOCOP III 是一个处理非线性约束优化问题的很有前途的工具。但是有许多问题仍然需要更进一步的注意和实算。这包括对比率ρ重要性的研究；注意将某些线性约束作为非线性约束表达是可能的；输入文件的改变将使参照点的空间更小，使线性可行搜索点的空间更大。不过，不清楚的是：这些变化是如何影响系统的执行的。

另一组实算是和一单个参数替换概率 p_r 相关的。在上述执行的所有实算中，p_r= 0.15。还有，我们计划在不久的将来扩展 GENOCOP III 以处理非线性等式。这要求一个另外的参数（ε）以定义系统的精度。所有的非线性等式 $h_j(\vec{X}) = 0$ （对 $j = q+1,\ldots,m$）将被一对不等式替换：

$$-\varepsilon \le h_j(\vec{X}) \le \varepsilon$$

这种新版本的 GENOCOP III 将能直接处理问题 G4。

① 这不是全局最优解；Bilchev[43]报告了 0.8348 的目标函数。

第8章 演化策略和其他方法

演化策略(ESs, Evolution Strategies)是一类仿效自然界进化规律以解决参数优化问题的方法[18,348]。它是60年代在德国发展起来的。如文献[348]中所描述的:

> "在 1963 年，两个德国柏林技术大学的学生碰到一起，并不久合作于流体工程学院的风洞实验。在流动中优化特体形状的研究中，一开始孕育的思想是策略性的。但试图用坐标和简单的梯度法没有成功的。然后，其中一个学生，Ingo Rechenberg，现在为仿生学和演化工程（Bionics and Evolutionary Engineering）的教授，偶然想到一个主意，即试着按照自然变异的实例随机改变定义形状的参数。演化策略于是就诞生了。"

（第二个学生是 Hans–Paul Schwefel，现在为计算机科学教授及系统分析主席。）

早期的演化策略可以被看成使用浮点数表达，只使用变异作为其重组算子的演化程序。它们已经应用于各种带连续可变参数的优化问题。只是在近来，它们被扩展到各种问题上[18,178]。

本章将讨论早期的基于两成员群体和变异算子的演化策略，以及各种多成员群体演化策略（8.1 节）。8.2 节比较了演化策略和遗传算法，8.4 节给出了近来由各个研究者建议的其他策略。

8.1 演化策略的进展

最早的演化策略只是基于由一个个体组成的群体。在演化过程中只使用了一个遗传算子：变异。不过，有趣的思想（在遗传算法中没用到）是一个个体被表达成一对浮点值向量，即 $\boldsymbol{v} = (\boldsymbol{x}, \boldsymbol{\sigma})$。这里，第一个向量 $\boldsymbol{x}$ 表示搜索空间的一个点；第二个向量 $\boldsymbol{\sigma}$ 是标准偏差向量。变异是通过对 $\boldsymbol{x}$ 的替换实现的：

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^{t} + N(0,\boldsymbol{\sigma})$$

这里 $N(0,\boldsymbol{\sigma})$ 为一个独立的带有零平均及标准偏差 $\boldsymbol{\sigma}$ 的随机 Gauss 向量。这是与对生物界的观察相一致的：较小的变化比较大的变换更经常地发生。后代当且仅当（被变异个体）有较好的适应值及被所有的约束（如果有）满足，它才被接受为群体中的一个新成员。例如，如果 f 为不带约束的求最大目标函数，当且仅当后代 $f(\boldsymbol{x}^{t+1}) > f(\boldsymbol{x}^{t})$，后代 $(\boldsymbol{x}^{t+1},\boldsymbol{\sigma})$ 替换其亲体 $(\boldsymbol{x}^{t},\boldsymbol{\sigma})$。否则，后代被消去，群体保持不变。

下面说明这样的演化策略的单一步骤。考虑一求最大问题，我们使用第 2 章中的例子：

$$f(x_1,x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

其中，$-3.0 \le x_1 \le 12.1$ ，$4.1 \le x_2 \le 5.8$。

如前所示，群体是由一单个个体$(\boldsymbol{x},\boldsymbol{\sigma})$组成，其中$\boldsymbol{x}=(x_1,x_2)$为搜索空间（$-3.0\le x_1\le 12.1$及$4.1\le x_2\le 5.8$）里的一个点，$\boldsymbol{\sigma}=(\sigma_1,\sigma_2)$表示两个用于变异算子的标准偏差。假定在某时刻$t$，此单成员群体由下面的个体组成：

$$(\boldsymbol{x}^t,\boldsymbol{\sigma})=((5.3,4.9),(1.0,1.0))$$

变异将产生下面的变化：

$$x_1^{t+1}=x_1^t+N(0,1.0)=5.3+0.4=5.7$$
$$x_2^{t+1}=x_2^t+N(0,1.0)=4.9-0.3=4.6$$

因为

$$f(\boldsymbol{x}^t)=f(5.3,4.9)=18.383705<24.849532=f(5.7,4.6)=f(\boldsymbol{x}^{t+1})$$

且x_1^{t+1}和x_2^{t+1}都维持在它们的区间里，所以后代将替换其单成员群体中的父代。

尽管群体是由经历变异的单个体组成，上述的演化策略被称为“两成员演化策略”。理由是后代同父代进行竞争，而且在竞争阶段，群体存在（临时的）两个个体。

标准偏差向量$\boldsymbol{\sigma}$在演化过程中保持不变。如果该向量的所有分量都是相同的，即$\boldsymbol{\sigma}=(\sigma,\ldots,\sigma)$，则优化问题为规则的(regular)① 。对收敛定理(Convergence Theorem)的证明见[18]：

定理 8.1　收敛定理：对$\sigma>0$及$f_{opt}>-\infty$（最小化）或者$f_{opt}<\infty$（最大化）的一个规则的优化问题，存在

$$p\{\lim_{t\to\infty}f(\boldsymbol{x}^t)=f_{opt}\}=1$$

收敛定理指出：经过充分长的搜索时间，全局最优解将以概率 1 被找到；但是它并不能给出有关收敛率（覆盖指向最优点的距离与覆盖这一距离所需经过代数之商）的任何线索。对收敛率优化，Rechenberg 建议了一个“1/5 成功法则”：

对所有的变异，成功变异的比率φ应该是 1/5。如果φ大于 1/5，增加变异算子的方差；否则，减少其方差。

从 1/5 成功法则可得到两个优化过程中函数优化收敛率的结论（所谓的回廊模型 — corridor model 和球形模型 — sphere model；详见[18]）。每隔k代，该法则被应用一次（k是方法的另一个参数），1/5 成功法则是：

$$\sigma^{t+1}=\begin{cases}c_d\cdot\sigma^t & 若\varphi(k)<1/5\\ c_i\cdot\sigma^t & 若\varphi(k)>1/5\\ \sigma^t & 若p_s(k)=1/5\end{cases}$$

其中，$\varphi(k)$为最近k代变异算子的成功比率。$c_i>1$和$c_d<1$调整变异方差的增加率和减小率。Schwefel 在其实算中[348]使用了下面的值：$c_d=0.82,\ c_i=1.22=1/0.82$。

1/5 成功法则直觉的理由是增加搜索的效率：如果成功，搜索将“加大”步骤继续；

① 如果目标函数f是连续的，则优化问题是规则的，函数的域是闭集，对所有$\varepsilon>0$，域中所有内点的集有函数值与优化值之差小于ε的点集合是非空的。对所有x_0，所有函数值小于或者等于$f(x_0)$的点的集合（对最小化问题而言；对最大化问题则成相反的关系）为一个闭集。

如果不是，该步骤将缩短。但是，这种搜索可能导致某类函数的过早收敛——这就导致对该方法的改善：增加群体规模。

多成员演化策略不同于先前的两成员策略，主要不同是群体规模（*pop_size*> 1)。多成员演化策略的其他特征是：

- 群体中的所有个体有同样的交配概率(mating probabilities) ；
- 引入一重组算子概率（在遗传算法界被称为"均匀杂交"），其中的两个（随机选择的）亲体

$$(\mathbf{x}^1, \boldsymbol{\sigma}^1) = ((x_1{}^1,...,x_n{}^1), (\sigma_1{}^1,...,\sigma_n{}^1))$$
$$(\mathbf{x}^2, \boldsymbol{\sigma}^2) = ((x_1{}^2,...,x_n{}^2), (\sigma_1{}^2,...,\sigma_n{}^2))$$

产生一个后代,

$$(\mathbf{x}, \boldsymbol{\sigma}) = ((x_1^{q_1},..., x_n^{q_n}), (\sigma_1^{q_1},...,\sigma_n^{q_n}))$$

其中，对所有的 $i = 1,...,n$，$q_n = 1$ 或 $q_n = 2$ 具有相等的概率。

变异算子及σ的调整保持不变。

在两成员和多成员演化策略之间仍然存在一些相似性：它们两者都生成单个后代。在两成员策略里，后代和其亲体竞争。在多成员策略里，最差个体（在 *pop_size* +1 个个体之间，即原始的 *pop_size* 个个体加一个后代）被消去。有一种方便的符合法能更进一步精炼地表示演化策略：

对两成员演化策略，(1+1) – ES

对多成员演化策略，(μ +1) – ES

其中$\mu = pop_size$。

多成员演化策略进一步发展成熟即变成[348]：

$(\mu + \lambda)$ – ES 和 (μ, λ) –ES

这些策略的主要思想是允许控制参数（像变异方差）自适应，而不是用一些确定性的算法来改变它们的值。

$(\mu + \lambda)$ – ES 是对多成员演化策略$(\mu + 1)$ – ES 一个自然的扩展，其中μ个个体生成λ个后代。新的（临时的）$(\mu + \lambda)$个个体的群体通过对μ个个体的再一次选择而减少。另外在(μ, λ) –ES 中，μ个个体生成λ个后代$(\lambda > \mu)$，而且选择过程只从λ个后代的集合中选择μ个个体组成一个新群体。这样做，每个个体的生命被限制在一代里。这就允许 (μ, λ) – ES 对随时间移动的最优解问题或者目标函数是带噪声的问题执行得更好。

在$(\mu + \lambda)$ –ES 和 (μ, λ) –ES 中使用的算子合并了两级水平的知识：它们的控制参数σ不再是固定的，或者它们不再通过某些确定性算法（像 1/5 成功法则）来改变，但它们被并入个体的编码结构里，并经历了演化过程。为产生一个后代，系统分几个阶段运作：

- 选择两个个体，

$$(\mathbf{x}^1, \boldsymbol{\sigma}^1) = ((x_1{}^1,...,x_n{}^1), (\sigma_1{}^1,...,\sigma_n{}^1))$$
$$(\mathbf{x}^2, \boldsymbol{\sigma}^2) = ((x_1{}^2,...,x_n{}^2), (\sigma_1{}^2,...,\sigma_n{}^2))$$

并应用重组（杂交）算子。有两种类型的杂交：

(1)不连续型杂交，杂交后的新后代是：

$$(\boldsymbol{x}, \boldsymbol{\sigma}) = ((x_1^{q_1}, \ldots, x_n^{q_n}), (\sigma_1^{q_1}, \ldots, \sigma_n^{q_n}))$$

这里 $q_i = 1$ 或 $q_{opt} = 2$（所以每个组分都来自于第一或第二预选的亲体）。

(2)中间体杂交，杂交后的新后代是：

$$(\boldsymbol{x}, \boldsymbol{\sigma}) = (((x_1^1 + x_1^2)/2, \ldots, (x_n^1 + x_n^2)/2), ((\sigma_1^1+\sigma_1^2)/2, \ldots, (\sigma_n^1+\sigma_n^2)/2))$$

这些算子的每一个还可以全局模式被应用，即新的一对亲体被选择作为后代向量的每个组分。

- 对所获得的后代$(\boldsymbol{x}, \boldsymbol{\sigma})$ 执行变异：产生的新后代是$(\boldsymbol{x}', \boldsymbol{\sigma}')$，其中

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0, \Delta\sigma)}$$

$$\boldsymbol{x}' = \boldsymbol{x} + N(0, \boldsymbol{\sigma}')$$

其中，$\Delta\sigma$为这种方法的参数。

为改进 ES 的收敛率，Schwefel[348]引入了另外一个控制参数$\boldsymbol{\theta}$。这种新的控制校正变异。对至今已讨论的 ES（对每个 x_n 拌有σ_n），搜索得更完美的方向只能是沿着系统坐标的轴建立。现在，群体中的每个个体被表达成

$$(\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$$

重组算子和前面段落中讨论的算子相似，变异按照下面方式从$(\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$产生后代 $(\boldsymbol{x}', \boldsymbol{\sigma}', \boldsymbol{\theta}')$：

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0, \Delta\sigma)}$$

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + N(0, \Delta\boldsymbol{\theta})$$

$$\boldsymbol{x}' = \boldsymbol{x} + \vec{C}(0, \boldsymbol{\sigma}', \boldsymbol{\theta}')$$

其中$\Delta\boldsymbol{\theta}$是方法的另一个参数，$\vec{C}(0, \boldsymbol{\sigma}', \boldsymbol{\theta}')$表示独立的平均值为零和具有适当概率密度的一个随机 Gauss 数向量（详情见[348]或[18]）。

演化策略在数值域执行得很好，因为它们（至少是初始地）奉献于（实的）函数优化问题。它们使用适当数据编码结构（通过控制策略参数扩展的浮点向量）及相应于问题域的“遗传”算子，是演化程序的一种例子。

将遗传算法和演化策略比较是十分有趣的，如它们的不同及相似之处、它们的长处和短处。我们将在下面章节讨论这些问题。

8.2 演化策略和遗传算法的比较

演化策略和遗传算法最基本的不同是它们的应用领域。演化策略的开发是针对数值优化。它们采用了一特殊的带自适应步长大小$\boldsymbol{\sigma}$和倾角$\boldsymbol{\theta}$的爬山过程。最近，演化策略已经用于离散的优化问题[178]。而遗传算法是作为（一般目的的）自适应搜索技术而形成的，这种搜索技术按指数增长的比率分配平均值之上的模式。遗传算法能用在各种领域，（实）参数的优化只是它们应用中的一个方面。

出于这种理由，用数值函数演化策略和遗传算法的时间和精度性能作为比较基础是

不公正的。不过，演化策略和遗传算法都是演化程序的例子，对它们的不同之处和相似之处的一般性讨论确实是十分自然的。

演化策略和遗传算法的主要相似之处是这两种系统都保持潜在解的群体并使用了适者生存选择原理。但是，这些方法之间还有许多不同之处。

演化策略和经典的遗传算法的第一个不同是它们表达个体的方式。在已描述的几种系统中，演化策略操作于浮点向量，而经典的遗传算法操作于二进制向量。

遗传算法和演化策略的第二个不同是选择过程本身的不同。在演化策略的单个代里，μ个亲体产生的中间群体由重组和变异算子［对$(\mu+\lambda)-ES$］生成的λ个后代加上原始的μ个亲体［对$(\mu, \lambda)-ES$］组成，然后，选择过程通过从群体中移去最不适的个体以减小此中间群体的大小。返回的由μ个个体组成的群体即构成了下一代。在遗传算法的单个代中，选择过程从大小为 *pop_size* 的群体中选择 *pop_size* 个个体。个体选择被重复，一个强壮的个体有较好的机会被新群体选择多次。同时，即使是最差个体也有机会被选择。

在演化策略里，选择过程是确定性的：它从$\mu+\lambda$个［$(\mu+\lambda)-ES$］或λ个［$(\mu, \lambda)-ES$］个体中选择最好的μ个个体（无重复）。而在遗传算法里，选择过程是随机的，从 *pop_size* 个个体中选择 *pop_size* 个个体（有重复），选择的机会与个体的适应值成比例。一些遗传算法实际上使用了分级加权选择；但较强个体仍可以被选择几次。换句话说，演化策略中的选择是静态的、消亡性的、与代有关的［对$(\mu, \lambda)-ES$］。而遗传算法的选择是动态的、被保存的、飞鸽型的 (*on-the-fly*)（见第 4 章）。

选择和重组步骤的相对次序构成了遗传算法和演化策略的第三个不同：在演化策略里，选择步骤跟随着重组算子的使用，而在遗传算法里，这些步骤的次序是相反的。在演化策略里，一个后代是两个亲体杂交和进一步变异的结果。当$\mu+\lambda$（或λ）个个体组成的中间群体已准备好，选择过程减少其大小到μ个个体。在遗传算法里，我们首先选择一中间群体。然后应用遗传算子（杂交和变异）到一些个体上（按照杂交概率选择）及一些基因上（按照变异概率选择）。

演化策略和遗传算法的下一个不同是在演化过程中，遗传算法的再生殖参数保持常数（杂交概率、变异概率），而演化策略始终改变参数（$\boldsymbol{\sigma}$和$\boldsymbol{\theta}$）：它们和解向量 $\boldsymbol{x}$ 一起经历杂交和变异，因为一个个体被解释为$(\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$的三位一体。这是十分重要的——演化策略中控制参数的自适应对应于系统的局部微调。

演化策略和遗传算法在处理约束时也存在着不同。演化策略承担一组 $q \ge 0$ 个不等式，

$$g_1(\boldsymbol{x}) \ge 0, \ldots, g_q(\boldsymbol{x}) \ge 0$$

作为优化问题的一部分，如果在一些迭代过程中，一个后代不满足所有这些约束，那么此后代即被取消资格，即不被放到新群体中。如果这样的非法个体发生率很高，演化策略就调整它们的控制参数，即减少向量$\boldsymbol{\sigma}$的组分。遗传算法处理约束（在第 7 章已讨论过）的主要策略是对违反约束个体强加惩罚。理由是对强约束问题，我们不能只是消去不合法后代（遗传算法不调节它们的控制参数）——否则，算法将长时间地原地不动。

同时，各种解码器或者修补算法都因消耗太大而不能加以考虑，即构造一个好的修补算法可能与解决问题本身一样费力。罚函数技术有许多缺点，其中之一是与问题有关。

上述讨论说明演化策略和遗传算法涉及到的许多细节上是十分不同的。不过，走近细看演化策略和遗传算法在过去 20 年的发展，人们不得不承认，这些方法之间的鸿沟正变得越来越小。

让我们再次讨论一些围绕演化策略和遗传算法的论点，这一次从历史面貌来讨论。

很早以前，就有迹象表明遗传算法在数值应用中执行局部搜索时，存在一些困难（见第 6 章）。许多研究者实算了不同的表达式（灰码、浮点数）及不同的算子以改进基于遗传算法系统的性能。今天，遗传算法和演化策略之间的第一个不同已不再是问题了：多数应用于参数优化问题的遗传算法都使用浮点表达[78]、适当方式的适应算子（见第 4 章）。似乎遗传算法借用了演化策略中向量表达的思想。

我们的演化程序的实算结果给出了一有趣的评述：杂交和变异单独运用在演化过程中都不是满意的。这两个算子（或这两类算子）在为系统提供好的性能都是必需的。杂交算子在搜索空间里勘探希望区域是十分重要的，并与早期的（但不是过早）收敛有关：在许多系统里——特别是那些工作于丰富的数据编码结构（本书的第 III 部分）的系统，杂交率的降低会使它们的性能恶化。同时，这些系统使用的变异率十分高：GENETIC–2 系统（第 9 章）使用的变异率为 0.2。

在演化策略界，也得到相似的结论：作为一个序列，杂交算子被引入到演化策略里。注意，早期的演化策略只是基于变异算子，杂交算子是后来才合并进来的[348]。看起来遗传算法和演化策略的得分是一样的：演化策略界借用了遗传算法界中杂交算子的思想。

有关演化策略和遗传算法关系还有更有趣的论点。最近，其他一些杂交算子被同时引入到遗传算法和演化策略里[269, 270, 352]。两个向量 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 可以生成两个后代，$\boldsymbol{y}_1$ 和 $\boldsymbol{y}_2$，它们是其亲体的线性组合，即

$$\boldsymbol{y}_1 = a \cdot \boldsymbol{x}_1 + (1-a) \cdot \boldsymbol{x}_2$$
$$\boldsymbol{y}_2 = (1-a) \cdot \boldsymbol{x}_1 + a \cdot \boldsymbol{x}_2$$

这样的杂交：

- 在遗传算法界，被称为保证平均杂交(guaranteed average crossover)[77]（当 $a = 1/2$），或者算术杂交(arithmetical crossover)[269, 270]。
- 在演化策略界，被称为中间体杂交(intermediate crossover)[352]。

演化策略中控制参数的自适应在一些遗传算法中也有其对应物。总的来说，在遗传算法运行中很早以前就采纳了这种思想：ARGOT 系统[354]采用个体的自适应表达（见 8.4 节）。遗传算法中适应控制参数的问题被公认已有一段时间了[169, 77, 115]。很明显，对一特定问题，从一开始就找到遗传算法中好的设定参数不是一件轻而易举的事。曾出现过几种方法。一种方法[169]使用了一个监督遗传算法(supervisor genetic algorithm)来优化对某类问题“适当”的遗传算法参数。考虑的参数有群体规模、杂交率、变异率、代隙（在每代中群体被替换的百分比）、比例窗及选择方案（纯选择或者精华选择）。另一个方法[77]包括遗传算子适应概率：其思想是应用一算子的概率按照该算子产生个体的表观性

能好坏来按比例地改变。直觉的意图是当前工作“好”的算子应该是更经常地使用。在[115]中，作者实算了四个分配变异算子概率的方案：（1）恒定概率；（2）指数型减少；（3）指数型增加；（4）方案（2）和（3）的组合。

同样，如果回忆一下非均匀变异（在第 6 章中有描述），我们会注意到，此算子在演化过程中改变其行为。

让我们简单地比较一下基于遗传的演化程序的 GENOCOP（第 7 章）和演化策略。这两个系统都维持一潜在解群体并使用一些选择惯例来区分“好”和“坏”个体。两个系统都使用了浮点数表达。它们都提供了高精度（演化策略采用控制参数，GENOCOP 采用非均匀变异）。两个系统都能优美地处理约束：GENOCOP 利用线性约束的处理，演化策略利用不等式的处理。两个系统都容易地合并来自对方的“约束处理思想”。算子是相似的。但一个使用了中间体杂交，另一个使用了算术杂交。它们真的不同吗？

从演化策略的视角来进行演化策略和遗传算法的比较在 187 中有描述。

几年前[117]，演化规划（EP）技术（见 13.1 节有关原始的演化规划技术的叙述）通常是用来处理数值优化问题的。它们和演化策略十分相似：都使用了浮点表达，而且变异是关键算子。演化策略和演化规划的基本不同可以概述如下[19]：

- 演化规划不使用重组算子；
- 演化规划使用了概率选择（竞争选择），而演化策略为下一代选择最好的μ个个体；
- 在演化规划中，由目标函数获得适应值，对它们进行标度，并可能对它们强加某种随机交替变换；
- 对每个个体变异的标准偏差的计算作为其自身适应值线性变换的平方根。

有关用于数值优化的演化规划更多的信息及演化策略和演化规划技术的实算比较，读者可参考[19, 117, 121]。

8.3 多峰和多目标函数优化

在本书的多数章节里，我们描述的方法都是确定一个函数单个的全局最优解。但在许多实例中，一个函数可能有几个我们希望确定的最优解（多峰优化）或者有多个优化判据（多目标优化）。很明显，解决这类问题需要新的技术，我们将依次讨论它们。

8.3.1 多峰优化

在许多应用中，确定一给定函数所有的最优解可能是很重要的①。对这类多峰优化，有几个方法是基于演化技术的。

第一种技术是基于迭代：我们只是重复算法的几个运行。如[30]中所讨论的，如果所有的优化解有一相等的可能被找到，独立运行的次数应该是：

① 我们理解“所有的”最优指感兴趣的解，即所有高于某一阈值的解。

$$p\sum_{i=1}^{p}\frac{1}{i}\approx p(\gamma+\log p)$$

其中 p 为最优解数目，$\gamma\approx 0.577$ 为 Euler 常数，不幸的是，在多数现实世界问题中，最优解多半不等，因此独立运行的次数应该高得多。也有可能使用这种迭代方法的并行实现，其中几个子群体同时演化（以独立的、互不通讯的方式）

Goldberg 和 Richardson[162]描述了一种基于共享的方法：该方法允许构造不同串的稳定子群体（物种）——用这种方法，算法平行地研究了许多峰［文章还给出了对其他使用了好位方法(niche methods)和种成形(species formation)思想的方法极好的综述］。共享函数确定由于距个体某距离 *dist* 的邻居的存在而造成的该个体适应值的退化[162]①。共享函数 *sh* 被定义成具有下面性质距离的函数：

- 对所有的距离 *dist*，$0\le sh(dist)\le 1$
- $sh(0)=1$
- $\lim_{dist\to\infty} sh=0$

有许多共享函数满足上述条件。有一种是[162]中建议的：

$$sh(dist)=\begin{cases}1-\left(\dfrac{dist}{\sigma_{sh}}\right)^{\alpha} & 若\ dist<\sigma_{sh}\\ 0 & 若\ dist\ge\sigma_{sh}\end{cases}$$

其中σ_{sh}和α 为常数（对这些常数重要性的讨论见[162]）。

一个个体 x 的新（共享）适应值给出如下：

$$eval'(x)=eval(x)/m(x),$$

其中 $m(x)$返回的是对一特定个体 x 的好位(*niche*)计算：

$$m(x)=\sum_{y} sh(dist(x,y))$$

在上述公式中，群体中所有 y 之和包括串 x 本身，因此，如果全是由它自身的好位形成的串，那么它的适应值不减少［即 $m(x)=1$］。否则适应函数值的减少与其邻近点的距离和数目成比例。

它意味着，当许多个体属于同一邻位，它们就逐一增加一个共享计数，这样就逐一减少了它们的适应值。结果是这种技术限制了群体中特殊物种无控制的生长[154]。

最近，Beasley，Bull 和 Martin[30]描述了解决多峰函数优化的一种新技术（被称为：序贯好位），它避免了共享法的一些缺点（即由于适应共享计算带来的时间复杂性，群体大小与优化解的数目成比例）。所建议的算法同样使用了一个距离函数 *dist* 和一适应值函数 *eval*，它是基于下面的思想：一旦一个优化解被找到，评价函数可以被修正以消去（已经发现的）这个解，因为再发现相同的优化解是没有用处的。在某种意义上，接下来运行的遗传算法合并了先前运行发现的知识（和简单的迭代技术相对，其中的每次运行都从一随机产生的群体开始）。该算法的基本步骤是（取自[30]）：

(1)初始化：使修正后的适应函数和原始的适应函数同等看待。

(2)使用修正的适应函数运行遗传算法，在运行中保持最好个体的记录。

① 距离 *dist* 可以定义在基因型或者遗传实现型水平上，即定义在串上或者对它们的解释上。

(3)更新修正的适应函数以削弱最好个体附近区域[①]。生成新的适应函数。

(4)如果最好个体的原始适应值是有意义的（即超过了某个阀值），将它作为一个解。

(5)如果还没有找到所有的解，回到步骤2。

对该算法的详细的讨论和实算结果，读者可以参考[30]。

Spears[365]最近建议了另一种方法。建议的算法实现了共享思想和限制性交配思想。但精确距离(metic distance)的思想被放弃，并以标记(label)的概念取而代之：群体中的每个个体有一个标记（在[365]报告的实算中，一个标记是一 n 位的串，因此，标记可以代表 2^n 个子群体)。为解释建议算法的本意。让我们引用[365]中的话：

“假设我们有简单函数含两个峰的，一个峰是另一个峰的两倍高，进一步假设对每个个体我们允许一标签位，每个标签位随机地被初始化，所以，在运行的开始，我们使用大致规模相同的子群体。由于是随机取样，两个子群体能最终驻留在一个较高峰上，或者驻留在一较低峰上。但是，在某种情况下（由于随意取样），每个子群体将朝不同的峰前进。如果我们不用适应值共享，在较高峰上的个体将总是比在较低峰上的个体产生更多的后代，最终，在较低峰上的子群体将会消失。但用适应共享，较高的峰只能支持两倍于较低峰所能支持的个体数（因为高度只是两倍的差别）。……适应共享机制能动态调节觉察得到的适应值，以便两个峰都有相同的觉察得到的高度。结果是两个子群体都以一种稳定的形式生存。而且，限制性交配阻止两个峰之间的个体进行杂交，这种杂交常产生低适应值个体。”

有关实验的详细结果，读者可以参考[365]。

最近，Mahfoud[249]的文献比较了几种好点（niching）方法，它可以被分成两类：序贯的和并行的，并行法在群体中同时形成和维持好点；序贯法暂时寻找多个好点。结果表明并行的好点法比序贯法性能好，有关这种比较的细节和并行法优点的详细讨论见[249]。

8.3.2 多目标优化

对许多真实世界的决策问题，需要同时优化多个目标。如[182]中所叙述的：

“一种可能的方法是……使用持久利益最大化(long-run profit maximization)作为单一目标。初一看，这种方法显示出值得考虑的优点。特别是，持久利益最大化的目标非常具体，能够方便地使用，但它似乎宽广到足以包容大多数机构的基本目标。实际上，一些人趋向于觉得所有合法的目标可以转化成这一个。但是，这种过分简单化要非常小心！大量研究已经发现，不是利益最大化，而是结合其他目标的利益满足才是美国企业的特性。特别是，典型的目标可能是维持稳定的利益，增加（或者保持）一个企业的市场份额、生产多样化的产品、维持稳定的价格、改进工人的士气、保持生意的家族控制和增加企业的威望。这些目标可能是和持久利益最大化相容的，但这些关系如

① 算法的描述假定为求最大化问题。

此模糊，以致将它们合并成一个目标很不方便。”

这种多目标优化问题需要分离技术，这种技术和单目标的标准优化技术很不相同。非常明显，如果有两个目标要被优化，有可能对第一个目标找到一个最好解，而对第二个目标又找到另一个最好解。

可以很方便地将多目标优化问题所有潜在解分成支配解(dominated solution)和非支配（Pareto 优化）解。若解 x 是支配的，则存在一可行解 y 对所有的对等物都不比 x 差，即对所有的目标 $f_i\,(i=1,\ldots,k)$ ①：

$$f_i(x) \leq f_i(y)，其中 1 \leq i \leq k$$

如果一个解不被任何其他可行解支配，我们称之为非支配（或称 Pareto 优化）解。所有 Pareto 优化解可能都有一定的意义。最理想地，系统应该回头报告所有 Pareto 优化点集合。

有一些多目标优化的经典方法[369]。这包括目标加权法，将多目标函数 f_i 组合成一个总的目标函数 F：

$$F(x) = \sum_{i=1}^{k} w_i f_i(x),$$

其中权重 $w_i \in [0,1]$ 有 $\sum_{i=1}^{k} w_i = 1$。不同的权重向量提供不同的 Pareto 优化解。另一种方法是距离函数法，它将多目标函数组合成一个基于需求水平向量 y 的目标函数：

$$F(x) = \left(\sum_{i=1}^{k} |f_i(x) - y_i|^r\right)^{\frac{1}{r}}$$

通常 $r=2$（Euclid 度量）。

多目标优化在遗传算法界有一定的兴趣，1984 年，Schaffer[339]开发了 VEGA 程序 (Vector Evaluated Genetic Algorithm)，它是对 GENESIS 程序的扩展[166]，其中包括了多判据函数。VEGA 系统的主要思想是将群体划分成相等规模的子群体；每个子群体对于单个目标是“合理的”。对每个目标，选择过程是独立执行的，但杂交是跨越子群体边界执行的。并开发和研究了其他启发式方法（例如资源的重分配模式、杂种方案），用来降低系统朝任何目标都不是最好解的个体的收敛趋向。

最近，Srinivas 和 Deb[369]建议了一种新技术，非支配排序遗传算法 NSGA (Nondominated Sorting Genetic Algorithm)，它是基于个体的几层分级的。在选择执行前，群体根据非支配被排序：所有非支配个体被分成一类（带有一虚拟适应值，它比例于群体规模，用以为这些个体提供相等的再生潜力）。为维持群体的多样性，这些被分级的个体共享它们的虚拟适应值（见前一节）。然后，忽略这组分级的个体，考虑另一层非支配个体。该过程继续直到群体中的所有个体被分级。有关该系统全面的讨论及第一手的实算结果，读者可以参考[369]。

最近，Fonseca 和 Fleming[127]对多目标优化的演化算法进行了综述，包括了这两类技术（即将许多判据组合成一个目标函数并返回一个信号值的技术；基于 Pareto 优化并返

① 对最大化问题：否则用小于等于不等式替换大于等于不等式。

回一套值的技术）的问题，并确认了几个公开的研究。

8.4 其他演化程序

如前所述，经典的遗传算法对局部微调不是适合的工具。出于这个理由，遗传算法给出的对数值优化问题的解要更不精确，譬如，比起演化策略来，除非遗传算法中的个体表达从二进制变成浮点，而且演化系统提供特殊算子（像非均匀变异，第 6 章）。然而，在近 10 年，一些人试图（直接地或间接地）改进遗传算法的这种特征。

一个对遗传算法的有趣的修正，称为δ编码，最近由 Whitley 等人建议[400]。这种方案的主要思想是处理群体中的每个个体不是作为问题的潜在解，而是作为一个另外的（小）值（称为δ值），它被加到当前的潜在解上。简化的δ编码算法如图 8.1 所示。

```
procedure Delta Coding
    begin
    apply GA on level x
    save the best solution (x)
    while (not termination–condition) do
    begin
        apply GA on level δ
        save the best solution (δ )
        modify the best solution (x level):
        x ← x + δ
    end
end
```

图 8.1 一个简化的δ编码算法

δ编码算法在两个水平上应用遗传算法技术：问题潜在解的水平（水平 x）和（迭代阶段）δ变化水平（在δ水平）。在 x 水平，单独应用遗传算法找到的最好解被保存 (x) 并留作一个参考点。然后在水平δ，执行内遗传算法的几个迭代。这一水平的遗传算法执行的终止（即当遗传算法收敛）将产生最佳修正后的向量δ，它将更新 x 的值。经过更新，下一次迭代再进行。在迭代阶段遗传算法的每次应用将重新随机初始化 δ群体。当然为评价个体δ，我们评价 $x+\delta$。

原始的δ编码算法更复杂，因为它对位串操作。这样做，δ编码遵守了遗传算法的基础理论（因为每次迭代要运行一次遗传算法）。两种水平上的遗传算法的终止条件通过群体中最好和最差个体的 Hamming 距离来表达（如果 Hamming 距离不大于 1，算法终止）。另外，有一个变量 *len* 表示用于表达向量δ单组分的位数（实际上，只有 *len*−1 表示组分的绝对值；最后位保存值的符号）。如果来自δ水平的最好解产生一个向量

$$\delta=(0,0,\dots,0)$$

（即最好潜在解 x 不变化），变量 *len* 加 1（增加解的精度），否则减 1。注意δ编码使变异成为不必需的，这是由于群体在水平δ上对每次迭代重新初始化的缘故。

我们可以简化原始的δ编码算法（图 8.1 给出了这样的简化示意图），如果对两个向量 x 和 δ都用浮点数序列表达，我们还可以改进其精度和时间性能。

δ编码算法的一些思想很早就出现在文献中。例如，Schraudoph 和 Belew[347]建议了一个动态参数编码（Dynamic Parameter Encoding，DPE）方案，其中被编码个体的精度是动态可调的。在该系统中，解向量的每个组分被一固定长度的二进制串表达；但是当（在某些迭代）遗传算法收敛，解的最重要位被放弃（当然在保存后！），其余的位左移一个位置，并且引入一个新的位。在最不重要的位置上该新位通过对搜索空间的更细微的分区增加了精度。过程不断重复直到某些全局终止条件被满足。

重新初始化群体的思想在[153]中有讨论，其中 Goldberg 研究了使用小群体规模的遗传算法系统，每当遗传算法收敛（当然保存最好个体），该系统就重新初始化。这种方案（被称为序列选择 — serial selection）如图 8.2 所示。

```
procedure Serial Selection
    begin
    generate a (small) population
    while (not termination-condition) do
    begin
        apply GA
        save the best solution (x)
        generate a new population by transferring the best individuals
            of converged population and then generaing the
            remaining individuals randomly
    end
end
```

图 8.2 基于重新初始化群体的遗传算法

群体的重新初始化在个体中引入多样性，并对系统的性能有正面的效果[153]。

包括一个学习组分的一种建议，与演化策略相似。Grefenstette [169]建议用另一个监督遗传算法来优化控制参数（群体规模、杂交和变异率等）。Shaefer[354]讨论了 ARGOT 方案(Adaptive Representation Genetic Optimizer Technique，自适应表达遗传优化器技术)，该系统学习个体的最好内部表达。

最近，另一个有趣的演化程序建议了一种选择性演化策略（IRM, 免疫补员机制，immune recruitment mechanism）[35]，来作为实空间的优化技术（一个类似的优化 Hamming 空间函数系统，被称为 GIRM）。该方案组合了前面见到的向期望的方向指导搜索的思想（例如 tabu 搜索）。像所有的演化规划技术一样，后代是由当前群体产生。在经典的遗传算法中，这样的后代替换其父代。在演化策略中，后代与其父代竞争（早期的演化策略），或者后代与父代和其他后代竞争［$(\mu+\lambda)-$ES］，或者它与其他后代竞争［(μ,λ)–ES］。在 IRM 系统中，后代必须传递一个和其邻局的亲缘的附加测试。该测试检查它是否展示其靠近的邻居有充分的相似性。

总之，一个可接受的候选 k 将通过亲缘测试，如果

$$\sum_i m(k,i)\cdot f_i > T$$

其中 i 表示已经出现在群体中的不同物种，f_i 为物种 i 的密集度，$m(k,i)$为物种 k 和 i 的亲缘函数，T 为补员阀值。

IRM 方案通过只接收满足亲缘测试的个体来指导搜索，相似的思想由 Glover 在其

研究分散搜索(Scatter Search)和 Tabu 搜索时进行了公式化的表达[142,145]。分散搜索技术，像其他演化程序，维持一潜在解的群体（向量 x^i 被称为参考点）。这种方案通过加权的线性组合统一首选的参考点子集以产生试探点（后代），并选择最好成员成为新参考点源（新群体）。这里一个新的扭转是多杂交（被称为加权组合）的使用，其中几个（多于两个）亲体用于产生一个后代。在[145]中，Glover 通过将它和 Tabu 搜索 —— 一个限制新后代选择的技术（它要求内存来保存个体的以往集合）——组合扩展了分散搜索的思想[143,144]。分散/tabu 搜索算法的结构如图 8.3 所示。

```
procedure scatter/tabu search
begin
    t = 0
    initialize P(t)
    evaluate P(t)
    classify P(t)
    while ( not termination-condition) do
    begin
        t=t+1
        create R(t)
        evaluate R(t)
        select P(t) from R(t) and P(t-1)
        classify P(t)
    end
end
```

图 8.3 分散/tabu 搜索

经过初始化和评价后，分散/tabu 搜索算法将解 $\bar{X}_1,\ldots,\bar{X}_{pop_size}$ 的群体分成［“classify $P(t)$” 阶段］几个集合。其中包括：(1) 由整个过程中一些（固定）数目的最好解组成的中坚历史生成器集合 V；(2) 由当前排除在外解组成的 tabu 生成器集合 $T \subseteq V$；(3) 由最好的 $V-T$ 成员组成的被选择历史生成器集合 V^*；(4) 由最好成员 S 组成的被选择的当前生成器集合 S^*。分级步骤［classify $P(t)$］在算法以后的迭代阶段一直执行。

每次迭代，生成试探点集合 $R(t)$。这些试探点对应于群体 $P(t)$的后代：它们被评价后，其中一些被加入到新群体中［8.3 中的“select $P(t)$ from $R(t)$ and $P(t-1)$”］。

最近，David Fogel[126]应用演化规划的思想来解决实型值的连续优化问题[119]；这些扩展包括自适应独立方差和协方差矩阵优化过程。

Maniezzo 开发了粒性演化(granularity evolution)的概念[250]，其中算法允许目标函数取样和目标函数样本分解（即粒性 — granularity）同时演化。个体是变长的，其编码按照在染色体中指定的特定分解水平来被解码。

同时，Muselli 和 Ridella 研究了处理间隔的遗传算法概念（称为间隔遗传算法——Interval Genetic Algorithm）[293]。间隔遗传算法融合了遗传算法和模拟退火的思想：遗传算子为：杂交，从两个间隔产生一个新间隔；合并，从两个间隔交叉的亲体中产生一个后代；变异，为一更好点搜索间隔（即单亲）。

看来，在“最优的优化器”搜索中多数较有希望的方向多少蕴含在上述思想中。每一种方案都提供了一种新见解，它们可能对开发解决某一类问题的演化程序是有用的。

正如 Glover 所说[145]：

“结构组合的使用使以显著不同于（经典的）杂交操作的方式组合组成向量成为可能。用遗传算法集成这样方法可以打开新型搜索方案之门。加权和适应结构组合的实现可以迅速地被完成以开拓环境，而其中的杂交并没有明显的意义（或者很难有可行性的保证），这些事实启示我们这种集成搜索过程可能对调整遗传算法当前应用上的限制是有利的。”

要明白地理解这些，我们需要进入到下一章。



第三部分 演 化 程 序



第9章　运输问题

在第 7 章，我们比较了处理约束的不同遗传算法。看起来，对某一类特定问题，如运输问题，我们可以做得更好：可以使用更自然的数据编码结构（对运输问题是一个矩阵）和特定的适于矩阵运算的遗传算子。这样的演化程序比 GENOCOP 有威力得多：GENOCOP 能优化带线性约束的任何函数，而新的演化程序只能优化运输问题（这类问题恰好有 $n+k-1$ 个等式，其中 n 和 k 分别表示来源地数和目的地数；见下面有关运输问题的描述）。然而，你会非常有趣地看到，通过引入额外的与问题有关的特定知识到演化程序里，我们所能赢得的硕果。

9.1 节给出了线性运输问题的演化程序[①]，9.2 节给出了非线性运输问题的演化程序[②]。

9.1　线性运输问题

运输问题（如[387]所述）是已研究过的包含约束的最简单的组合问题之一。它确定一单个商品从一定数目的来源地到一定数目的目的地最小运输费用的方案。它要求给出每个来源地供货水平的规格、每个目的地需要的数量及从每个来源地到每个目的地的运输费用。

因为只有一种商品，一个目的地可以从一个或多个来源地接受其需求。目标是找到从每个起源地到每个目的地的装运数量以使整个运输费用最少。

如果在一条线路上的费用直接与运输量成正比，则运输问题是线性的；否则是非线性的。线性问题可以用运筹学(operations Research, OR)的方法解决，而非线性问题则缺乏一般的解决方法。

假定有 n 个来源地和 k 个目的地。来源地 i 供货的数量是 $sour(i)$，目的地 j 的需求量是 $dest(j)$。在来源地 i 和目的地 j 之间的单位运输费用为 $cost(i, j)$。如果 x_{ij} 为从来源地 i 到目的地 j 的运输量，那么运输问题可以写成：

$$\text{最小化}\quad \sum_{i=1}^{n}\sum_{j=1}^{k} f_{ij}(x_{ij})$$

并服从

$$\sum_{j=1}^{k} x_{ij} \leqslant sour(i),\quad i=1,2,\ldots,n,$$

$$\sum_{i=1}^{n} x_{ij} \geqslant dest(j),\quad j=1,2,\ldots,k,$$

① 重印部分得到 IEEE Transactions on Systems, Man, and Cybernetics, Vol.21.No.2, p445-452,1991 的许可。

② 重印部分得到 ORSA Journal on Computing, Vol. 3, No. 4, 1991, pp. 307—316, 1991 的许可。

$$x_{ij} \geq 0, \qquad i=1,2,\ldots,n \ , \ j=1,2,\ldots,k$$

第一组约束规定了一个来源地的总装运量不能超过其供货量；第二组约束要求到达某一目的地装运量之和必须满足其需求量。如果对所有的 i 和 j，$f_{ij}(x_{ij}) = cost_{ij} \cdot x_{ij}$，则问题是线性的。

上述问题意味着总的供货量 $\sum_{i=1}^{k} sour(i)$ 必须至少等于总的需求量 $\sum_{j=1}^{n} dest(j)$。当总的供货量等于总的需求量时，导出的公式被称为**平衡运输问题**，它与上面不同的只是所有相应的约束都是等式；即，

$$\sum_{j=1}^{k} x_{ij} = sour(i), \qquad i=1,2,\ldots,n$$

$$\sum_{i=1}^{n} x_{ij} = dest(j), \qquad j=1,2,\ldots,k$$

如果所有的 $sour(i)$ 和 $dest(j)$ 都是整数，则任何平衡线性运输问题的最优解是整数解，即所有的 x_{ij}（$i=1,2,\ldots,n, j=1,2,\ldots,k$）是整数，且 x_{ij} 中的正整数数目至多是 $k+n-1$ 个。在这一节，我们要解一个平衡线性运输问题。例子如下。有关运输问题及平衡方面的其他信息，读者可以参考有关运筹学的基础教材，见[387]。

例 9.1 假定 3 个来源地和 4 个目的地。供货量为：

$$sour(1) = 15，sour(2) = 25，sour(3) = 5$$

需求量为：

$$dest(1) = 5，dest(2) = 15，dest(3) = 15，dest(4) = 10$$

注意总供货等于总需求为 45。

单位运输费用 $cost(i, j)$ ($i=1,2,3$，$j=1,2,3,4$）在下表给出。

费 用

10	0	20	11
12	7	9	20
0	14	16	18

优化解如下所示。总费用为 315。解是由整数值 x_{ij} 组成。

运输量 x_{ij}

	5	15	15	10
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

9.1.1 经典遗传算法

我们这里"经典的"含义当然是使用位串染色体（即解的表达），由 0 和 1 排列成序列的遗传算法。在运输问题中定义解的位向量最直接的方法是产生一向量 $\langle v_1, v_2, \ldots, v_p \rangle$（$p = n \cdot k$），每个组份 v_i（$i=1,2,\ldots,p$）为一个位向量 $\langle w_0^i,\ldots,w_s^i \rangle$，并表达一个和分配矩阵的行 j 和列 m 相关连的整数。其中 $j = \lfloor (i-1)/k + 1 \rfloor$ 及 $m = (i-1) \bmod k + 1$（$\bmod k+1$ 以 $k+1$ 为模）。向量 w 的长度（参数 s）确定它可以表达的最大整数（$2^{s+1}-1$）。

让我们简要地讨论以上表述在满足约束、评价函数和遗传算子上产生的结果。

1. 约束满足

很明显，每个解向量必须满足下面条件：

- $v_q \geqslant 0$ 　　　　　$q=1,2,\ldots,k\cdot n$
- $\sum_{i=c\cdot k+1}^{c\cdot k+k} v_i = sour[c+1]$，$c=0,1,\ldots,n-1$
- $\sum_{j=m,step\ k}^{k\cdot n} v_i = dest[m]$，　$m=1,2\ldots,k$

注意，第一个约束总是满足的（我们把 0 和 1 的序列解释为一正整数）。其他两个约束给出每个来源地总和及每个目的地的总和，这些公式是不对称的。

2. 评价函数

自然的评价函数表示从来源地到目的地总的运输项费用，并由下面公式给出：

$$eval(\langle v_1,v_2,\ldots,v_p\rangle)=\sum_{i=1}^{p} v_i\cdot cost[j][m]$$

其中 $j=\lfloor(i-1)/k+1\rfloor$ 及 $m=(i-1)\ mod\ k+1$。

3. 遗传算子

对使用上面表达的运输问题，没有自然定义的遗传算子。变异通常被定义为解向量中单个位的变化。这将对应于一个整数值 v_i 的变化。这样，我们的问题将反过来触发一系列在不同位置上的变化（至少三个另外的变化）以维持约束等式。还要注意我们总是不得不记住在哪一行和哪一列发生了变化——尽管我们考虑并操作的向量表达是用行和列表示的（来源地和目的地）。这就是公式十分复杂的理由：这种复杂性的第一个信号是在表达约束时对称性的失去。

还有一些其他的公共的问题。变异被理解为在一解向量里的最小变化，但如我们先前注意到的，一个整数单个的变化将触发相应位置上至少三个其他的变化。假定两个随机点（v_i 和 v_m，其中 $i<m$）被选择并且它们不属于同一行或者列。假定 v_i，v_j，v_k，v_m（$i<j<k<m$）为一解向量的组份（被选择变异），并使 v_i 和 v_k 以及 v_j 和 v_m 属于同一列，v_i 和 v_j 以及 v_k 和 v_m 属于同一行。

即在矩阵表达中：

$$\begin{pmatrix}
\cdots & \cdot & \cdots & \cdot & \cdots\\
\cdots & \cdot & \cdots & \cdot & \cdots\\
\cdots & v_i & \cdots & v_j & \cdots\\
\cdots & \cdot & \cdots & \cdot & \cdots\\
\cdots & \cdot & \cdots & \cdot & \cdots\\
\cdots & v_k & \cdots & v_m & \cdots\\
\cdots & \cdot & \cdots & \cdot & \cdots\\
\cdots & \cdot & \cdots & \cdot & \cdots
\end{pmatrix}$$

现在，在试着确定解向量中最小的变化时，我们遇到了困难。我们应当增加还是减少 v_i 呢？可以选择用 1 来改变它（最小可能的变化），或者通过在范围$\langle 0,1,\ldots,v_i\rangle$里的

一些随机数。如果用某个常数 C 增加值 v_i，我们就不得不用同一量减小值 v_j 和 v_k。如果 $v_j < C$ 或者 $v_k < C$，将会有什么发生呢？可以设定 $C = \min(v_i, v_j, v_k)$，但那样的话多数变异将会导致无变化，因为选择三个非零成员的概率接近零（对长度为 n^2 的向量，小于 $1/n$）。

这种涉及单个位变化的方法将使变异算子对被选择成员对应的行或者列的复杂表达没有效率。

如果我们试图在应用杂交算子后修补一个染色体，则情形甚至更加复杂。在一随机点打断一个向量将导致一对染色体违反大量的约束。如果试图修正这些解使它们满足所有的约束，它们又将失去同亲体的多数相似性。而且，这样做的方式还很不明确：如果向量 **v** 在搜索空间之外，“修补”它可能和解原始问题一样困难。尽管我们可以成功地建立一个基于修补算法的系统，这种系统将会有较高的问题相关性，难以成为一般性的系统。

我们的结论是，上述向量表达不太适合定义这类约束问题的遗传算子。

9.1.2 引入与问题有关的知识

可不可以改进解的表达，同时又保持这个向量表达的基本结构呢？我们相信可以，但必须在表达中引入与问题有关的知识。

首先，让我们描述一种方式来产生一个满足所有约束的解，我们称这个步骤为*初始化*。当我们讨论两维结构的遗传算子时，它是变异算子的基本组份。它生成一个最多有 $k+n-1$ 个非零元素的矩阵，以便使所有的约束都被满足。该算法的结构如下，我们用例 9.1 的矩阵来解释它。

input: array $dest[k]$, $sour[n]$;
output: an array $(v)_{ij}$ such that $v_{ij} \geq 0$ for all i and j,

$$\sum_{j=1}^{k} v_{ij} = dest[i] \text{ for } i=1,2,\ldots,n, \text{ and}$$

$$\sum_{i=1}^{n} v_{ij} = sour[j] \text{ for } j=1,2,\ldots,k,$$

i.e., all constraints are satisfied.

```
procedure initialization;
begin
    set all numbers from 1 to k·n as unvisited
    repeat
        select an unvisited random number q
            from 1 to k·n and set it as visited
        set (row) i = ⌊(q−1) / k + 1⌋
        set (column) j = (q−1) mod k + 1
        set val = min(sour[i], dest[j])
        set v_ij = val
        set sour[i] = sour[i] − val
        set dest[j] = dest[j] − val
    until all numbers are visited.
end
```

例 9.2

例 9.1 的矩阵，即，

$$sour[1] = 15,\ sour[2] = 25,\ sour[3] = 5$$

$$dest[1] = 5,\ dest[2] = 15,\ dest[3] = 15,\ dest[4] = 10$$

合起来有 3×4 = 12 个数，它们中的全体在开始未被访问过。选择第一个随机数，譬如说 10。移动该数位于行 i=3 及列 j=2 上。$val = \min(sour[3], dest[2]) = 5$，所以 $v_{32} = 5$。还要注意在第一次迭代后，$sour[3] = 0$ 及 $dest[2] = 10$。

我们用下面三个（未被访问过）随机的数重复这些计算，设为 8、5 和 3（分别对应于行 2 和列 4、行 2 和列 1 及行 1 和列 3）。到目前为止，导出最终的矩阵 v_{ij} 为：

	0	10	0	0
0			15	
10	5			10
0		5		

注意 $sour[i]$ 和 $dest[j]$的值经过 4 次迭代给出。如果更进一步的随机数序列为 1, 11, 4, 12, 7, 6, 9, 2。产生的最终矩阵（按假定的随机数序列〈10, 8, 5, 3, 1, 11, 4, 12, 7, 6, 9, 2〉）为：

	0	0	0	0
0	0	0	15	0
0	5	10	0	10
0	0	5	0	0

很明显，经过 12 次迭代后，所有的（局部拷贝）$sour[i]$及 $dest[j] = 0$。还要注意，有几个数的序列，过程 **initialization** 将为此产生优化解。例如优化解（在例 9.1 中给出）可以从下面序列中的任一个：〈7, 9, 4, 2, 6, *, *, *, *, *, *, *〉及许多其他序列中获得，其中*表示任何未被访问的数。

这种技术可以产生包括至多 k+n−1 个非零整数元素的任意可行解。它将不产生其他虽可行但不享有这种特征的解。当试图解决运输问题的非线性版本时，初始化过程肯定必须要修正。

注意下面的矩阵：

	5	15	10	10
15	4	7	4	0
25	1	6	10	8
5	0	2	1	2

没有任何相对应的数序列。

似乎，对运输问题的优化解总是至多由 k+n−1 个非零元素组成，并可以由过程 **initialization** 生成。所以，我们不必关注解（如先前例子结尾给出的解），因为它没有对应的数序列。

这些有关问题的知识和其解的特征给我们另一个机会将运输问题的解表达成一个向量。一个解向量是一个在范围〈1, $k\cdot n$〉里的 $k\cdot n$ 个不同的整数组成的一个序列。它根据过程 **initializiaton** 产生一可接受的解。换句话说，我们将把一个解向量看成是数的排列，而

且将寻找对应此优化解的特殊排列。

下面简要地讨论该表达在约束满足、评价函数和遗传算子上的实现。

约束满足 任何有 $k \cdot n$ 个不同的数的排列产生满足所有约束的唯一解。这是通过过程 **initialization** 得以保证的。

评价函数 这相当容易：任何排列将对应于一个唯一的矩阵，设为 (v_{ij})。评价函数为

$$\sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij} \cdot cost[i][j]$$

遗传算子 这也是直接了当的：

- 倒置：任何解向量 $\langle x_1, x_2, ..., x_q \rangle (q = k \cdot n)$ 可以很容易地颠倒成另一个解向量 $\langle x_q, x_{q-1}, ..., x_1 \rangle$
- 变异：一解向量 $\langle x_1, x_2, ..., x_q \rangle$ 的任何两个元素，设为 x_i 及 x_j，可以被很容易地被交换而产生另一个解向量。
- 杂交：这有一点复杂。注意一任意（盲目）的杂交算子将导致非法解：应用下面的杂交算子到序列里

  $\langle 1, 2, 3, 4, 5, 6, | 7, 8, 9, 10, 11, 12 \rangle$ 及

  $\langle 7, 3, 1, 11, 4, 12, | 5, 2, 10, 9, 6, 8 \rangle$

  将产生（这里，杂交点是在第 6 位置之后）

  $\langle 1, 2, 3, 4, 5, 6, 5, 2, 10, 9, 6, 8 \rangle$ 及

  $\langle 7, 3, 1, 11, 4, 12, 7, 8, 9, 10, 11, 12 \rangle$

  它们中没有一个是合法解。

这样，我们不得不使用某些形式的启发式杂交算子。这些解向量序列和货郎担问题的解序列（见[170]）有一些相似之处。这里，我们使用了一个启发式杂交算子（属于 PMX 杂交算子家族，见[160]和第 10 章），对给定的两个亲体，下面的步骤将产生一个后代：

(1) 对第二个亲体做一个拷贝；

(2) 从第一个亲体选择任一部分；

(3) 在后代里做最小的变化以完成所选模式。

例如上述例子中的亲体，被选择部分为

$$(4, 5, 6, 7)$$

产生的后代是

$$\langle 3, 1, 11, 4, 5, 6, 7, 12, 2, 10, 9, 8 \rangle$$

正如所要求的，后代继承两个亲体的结构关系。亲体的角色在构造第二个后代时可以颠倒。

遗传系统 GENETIC-1 是基于上述的原理而创建的。它的实算结果将在下一节讨论。

9.1.3 作为表达结构的矩阵

对运输问题也许多数解的自然表达是两维结构。毕竟，在手工计算时问题就是这样被表达和解决的。换句话说，一个矩阵 $V=(v_{ij})$ （$1\le i\le k, 1\le j\le n$）可以表达一个解。

下面讨论矩阵表达在约束满足、评价函数和遗传算子上的实现。

约束满足 很清楚，每个解矩阵 $V=(v_{ij})$ 应该满足下面的条件：

- $v_{ij}\geqslant 0$， $i=1,\ldots,k,\quad j=1,\ldots,n$
- $\sum_{i=1}^{k} v_{ij} = dest[j] \quad j=1,\ldots,n$
- $\sum_{j=1}^{n} v_{ij} = sour[i] \quad i=1,\ldots,k$

这和直接法的约束集相似（见9.1.2节），但是约束以一种更简单更自然的方式表达。

评价函数 自然的评价函数通常是目标函数：

$$eval(v_{ij}) = \sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}\cdot cost[j][m]$$

该公式还是比直接法简单得多，且运行时快于系统 GENETICS-1，其中的每个序列在评价前必须被转变为（初始化）解矩阵。

遗传算子 我们在这里定义了两个遗传算子：变异和杂交。在这种特例下难以定义有意义的倒置算子。

- 变异：假定 $\{i_1,i_2,\ldots,i_p\}$为$\{1,2,\ldots,k\}$的子集，且$\{j_1,j_2,\ldots,j_q\}$为$\{1,2,\ldots,n\}$的子集，有$2\le p\le k, 2\le q\le n$。

我们以（$k\times n$）矩阵 $V=(v_{ij})$表示变异的一个亲体。那么，可以从矩阵 V 的全体元素按照下面的方式产生一$(p\times q)$子矩阵 $W=(w_{ij})$：元素 $v_{ij}\in V$ 并在是 W 中，当且仅当 $i\in\{i_1, i_2,\ldots,i_p\}$及 $j\in\{j_1,j_2,\ldots,j_q\}$（如果 $i=i_r$ 及 $j=j_s$，那么元素 v_{ij} 被放置在矩阵 W 的第 r 行、第 s 列）。

现在，可以为矩阵 W 分配新的值 $sour_W[i]$和 $dest_W[j]$（$1\le i\le p, 1\le j\le q$）：

$$sour_W[i] = \sum_{j\in\{j_1,j_2,\ldots,j_q\}} v_{ij}, \quad 1\le i\le p$$

$$dest_W[j] = \sum_{i\in\{i_1,i_2,\ldots,i_p\}} v_{ij}, \quad 1\le j\le q$$

我们可以使用过程 **initialization**（见9.1.3小节）来为矩阵 W 分配新值，使所有的约束 $sour_W[i]$和 $dest_W[j]$ 都被满足。此后，用矩阵 W 中的某个新元素替换矩阵 V 的适当元素。按照这种方式，所有的全局约束（$sour[i]$ 及 $dest[j]$）都被维持。

下面的例子将说明变异算子。

例9.3 给定一个有4个来源地和5个目的地及下面约束的问题：

$sour[1]=8, sour[2]=4, sour[3]=12, sour[4]=6,$
$dest[1]=3, dest[2]=5, dest[3]=10, dest[14]=7, dest[5]=5$

假设下面的矩阵 V 被选择作为变异的一个亲体：

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

随机地选择两行{2,4}和三列{2,3,5}。相应的子矩阵 W 为：

4	0	0
1	0	2

注意，$sour_W[1]=4$, $sour_W[2]=3$, $dest_W[1]=5$, $dest_W[2]=0$, $dest_W[3]=2$。经过矩阵 W 的重新初始化后，矩阵可能得到下面的值：

2	0	2
3	0	0

所以经过变异后，最终矩阵 V 的后代为：

0	0	5	0	3
0	2	0	0	2
0	0	5	7	0
3	3	0	0	0

- 杂交：

假定两个矩阵 $V_1=(v_{ij}^1)$ 和 $V_2=(v_{ij}^2)$被选择作为杂交操作的亲体。下面我们描述用来生成一对后代 V_3 和 V_4 算法的骨架。

生成两个临时矩阵：$DIV=(div_{ij})$ 和 $REM=(rem_{ij})$。它们定义如下：

$$div_{ij}=\lfloor (v_{ij}^1+v_{ij}^2)/2 \rfloor$$

$$rem_{ij}=(v_{ij}^1+v_{ij}^2) \bmod 2$$

矩阵 DIV 保持原来两个亲体的圆整平均值，矩阵 REM 保存对是否圆整是必须的。矩阵 REM 有一些有趣的性质：每行和每列的 1 的个数是偶数。换句话说，$sour_{REM}[i]$ 和 $dest_{REM}[j]$的值（分别为矩阵 REM 行和列的边和）是偶数。我们利用这种性质来将矩阵 REM 变形成两个矩阵 REM_1 和 REM_2，并使

$$REM=REM_1+REM_2,$$

$$sour_{REM_1}[i]=sour_{REM_2}[i]=sour_{REM}[i]/2, \qquad i=1,\dots,k,$$

$$dest_{REM_1}[j]=dest_{REM_2}[j]=dest_{REM}[j]/2, \qquad j=1,\dots,n.$$

然后，我们产生 V_1 和 V_2 的两个后代：

$$V_3=DIV+REM_1$$

$$V_4=DIV+REM_2$$

下面的例子将说明这种方式。

例 9.4 采用和例 9.1 描述的相同的问题。假定下面的矩阵 V_1 和 V_2 被选择作为杂交的亲体：

V_1

1	0	0	7	0
0	4	0	0	2
2	1	4	0	5
0	0	6	0	0

V_2

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

矩阵 DIV 和 REM 为：

V_1

0	0	2	3	1
0	4	0	0	0
1	0	4	3	2
1	0	3	0	1

V_2

1	0	1	1	1
0	0	0	0	0
0	1	1	1	1
1	1	0	0	0

两个矩阵 REM_1 和 REM_2 为：

V_1

0	0	1	0	1
0	0	0	0	0
0	1	0	1	0
1	0	0	0	0

V_2

1	0	0	1	0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0

最后，两个后代 V_3 和 V_4 为：

V_1

0	0	3	3	2
0	4	0	0	0
1	1	4	4	2
2	0	3	0	1

V_2

1	0	2	4	1
0	4	0	0	0
1	0	5	3	3
1	1	3	0	1

基于上述原理，我们建立了一个演化系统 GENETIC–2。并按照第一种格调进行了实算，然后比较了修正的基于向量的经典的 GENETIC–1 和基于矩阵的更迭的 GENETIC–2 版本用在解决标准的线性运输问题的结果。

当然，我们的意图不是比较遗传方法和标准的优化算法。对我们挑选的每个有效测度，遗传方法是无法竞争的。如果我们把这种方法用到非线性的特例时，情形则会不同。通过使用一些不同大小并已知解的问题，我们意在研究问题表达的效果（GENETIC–1 对 GENETIC–2）。

一些随机产生的人工问题和一些已出版的例子包含了测试问题。人工问题随机产生单位费用、供货和需求值，尽管问题依然是平衡问题。我们感觉已出版的例子比人工问题包含更典型的费用结构。例如，产品清单(production–inventory)问题，当表达成一个运输问题时，有公认的费用模式，

对每种特例，问题首先使用标准的运输算法来解，以便用知道的最优值作为后来进行技术比较的终止判据。

问题的大小受我们使用的计算机的限制（Macintosh SE/30、Macintosh II、AT&T 3B2 及 Sun 3/60 ，后两个在 UNIX 操作系统下运行）。所引用的问题列在表 9.1 中。

表 9.1　使用的问题

问 题 名 称	大　小	引 用 文 献
prob01~prob15	4 乘 4 到 10 乘 10	问题随机产生
sas218	5 乘 5	[338], p.218
taha170	3 乘 4	[387], p.170
taha197	5 乘 4	[387], p.197
win268	9 乘 5	[405], p.268

在比较优化算法时，首先必须决定所使用的准则。一个显而易见的准则是到达最优

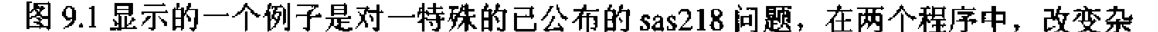
值所要求的代数，也许可以同时要求完成每一代消耗的时间或者操作次数。严重的问题是，在某种特例下，遗传算法需要许多代才能到达一个最优解。况且对参数的某些设定，可能在运行停止前根本没有最优解。所需的代数也可能随着随机数开始使用的种子及被解决的问题而有显著的变化。我们感到这种测度尽管是自然的，但对这些特定的实算却并不是容易使用的。

可供选择的、也是更实用的准则是在一固定的代数里到达最优值的靠近度。我们选择在 100 代里得到高于已知最优值的百分数。对 1000 代我们也做了观察，但在多数特例下，对研究的问题，解是到达还是非常接近最优值对此算法和被比较的算法都是没有结论的。

其他研究者已经发现（参见[169]）改变遗传算法的参数可以产生不同的执行效果。我们首先处理两种方法调节参数的实算结果。保持群体规模在 40，每代中选择再生数目固定在 10（群体的 25%）。后一个数字也是在每代中移走解的数目。于是就能调整变异参数，*cross*、*inv* 及 *mut* 的值，即，由杂交、倒置和变异挑选再生的亲体数分别被保持在总和为 10。杂交的数目（cross）必须是偶数，因为杂交必须在成对的亲体之间发生。倒置是仅有基于向量版本 GENETIC–1 可能有的。我们同样固定概率分布参数 *sprob*，它控制挑选亲体和挑选被移去个体的几何分布。

在调节过程中，进行了超过 1000 次的运行。对每个被挑选参数组合，运行是在五个不同的随机数种子下进行的。五次运行的平均目标值被转化为高于已知最优解的百分数。

图 9.1 显示的一个例子是对一特殊的已公布的 sas218 问题，在两个程序中，改变杂



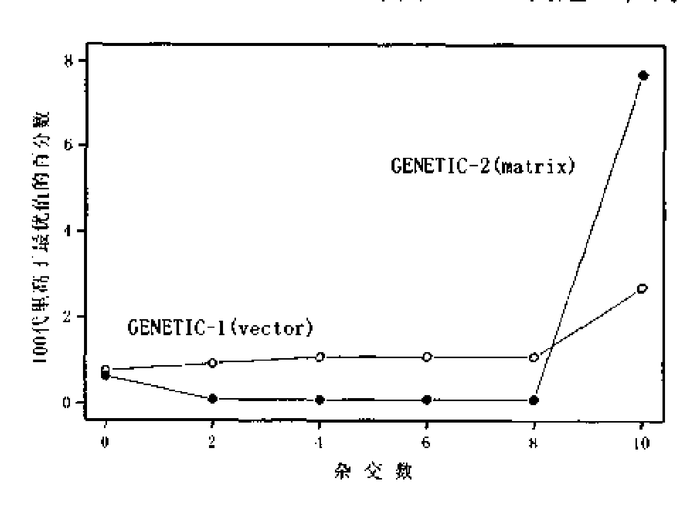


图 9.1 解问题 sas218 的两个算法

交对数目 *cross* 所产生的效果。相似地，对其他已公布的问题和人工问题，尽管不认同，但还是找到了结果。因为我们固定亲体的总数，较多的杂交意味着几乎没有变异，对向量模型，则是几乎没有倒置。图中的每个点是以不同的开始种子五次运行的平均值。在 100 代里，高于已知最优值的百分数在图中进行了绘制。

总之，基于矩阵的 GENETIC–2 版得到了作为杂交对数目函数的更平滑的曲线。通常，杂交越少，曲线越好。最好结果发生在零杂交对。但结果受挑选问题的影响很大。

问题 sas218 不同于多数其他问题， 在于最好结果是用 2 到 4 个杂交对获得的。GENETIC–1 的结果展示，通常增加杂交百分比率不是普遍适用的规律。对双方来说，如图中所展示的那样，当所有的对都被用于杂交方式、而不进行随机变异，形势就会恶化。GENETIC–2 模型对这种效果特别敏感，在这种情况下，它的性能甚至要比 GENETIC–1 差。

对研究的这类问题，我们发现尽管结果随问题而不同，小比例的杂交对两种模型都工作得很好。对向量模型，零倒置最好。

一旦获得了最优调节参数，我们就可以进行用运行收集的问题来比较模型。再次引用的结果都是每个特例五次不同开始种子下运行的平均值。

图 9.2 展示了在整个问题集的两个杂交对对问题大小（$n*k$）的曲线。在每种特例下，基于矩阵的 GENETIC–2 比基于向量的 GENETIC–1 执行的更好些（即，在 100 代里，更靠近最优解）。对同一问题，向量版从来没有比矩阵版执行得好。GENETIC–1 也比 GENETIC–2 更不可靠。对一些问题，这种效果特别引人注目，这可以从图中的分离点看出。

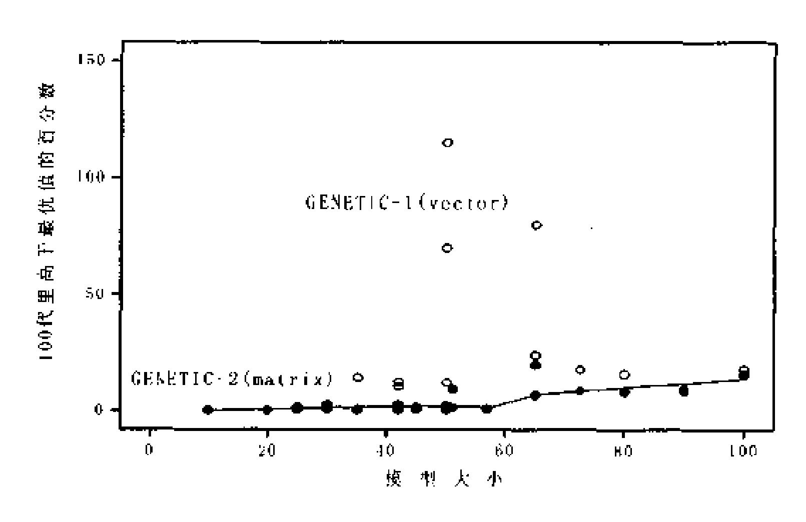


图 9.2 GENETIC–1 与 GENETIC–2 的比较

9.1.4 结论

我们的结论是：在这些实算里，使用我们的准则，基于矩阵的算法(GENETIC–2)比基于向量的算法（GENETIC–1）执行的更好。注意，这种比较是在基于矩阵的版本和特意开发的基于向量的模型之间开展的。基于矩阵的算法以自然的样式包含与问题有关的知识，而基于向量的模型为便于处理也必须依赖于附加的假设：一个特殊的基于详细的对问题的分析初始化步骤(**initialization**)，一定要设计出来以使基于向量的版本能完全地工作。由此，特殊的基于向量的版本(GENETIC–1)不能很容易地一般化，但矩阵方法(GENETIC–2)对一般化是具有潜力的，会有很好的前途，它甚至能包括更复杂的运输问题的非线性版本。这时 GENETIC–1 不能适当地工作：作为该系统的基础的过程 **initialization**，在线性情况下严重地依赖于的有关解形式的知识。当最优解不是整数矩阵且非零数可以大于 $k+n-1$ 很多时，GENETIC–1 系统必定失败。尽管我们改变了

initializaiton 过程 （例如，对每个被选择的第 i 行和第 j 列，变量 *val* 被分配一个在范围 $\langle 0, \min(sour[i], dest[j])\rangle$ 里的一个随机数），一个代表初始化点序列的向量应该扩展以便还能记录所有被选择的随机数。我们的结论是所有这些困难是由于解的人工表达是一个向量造成的。

遗传算法很慢，无法和基于标准线性规划算法的特定优化技术相比。后者按照问题大小（$n * k$）的次序在一定数目的迭代里能解决问题，而每一种遗传方法的每代包括构造一组对问题的潜在解。但它拥有的潜力对非线性和固定收费问题还是很有用的，而对标准的运输方法却显得无能为力（见下一节）。

9.2 非线性运输问题

在这一节，我们将用五个遗传算法的组成部分来讨论平衡非线性运输问题的演化程序：表达、初始化、评价、算子及参数。算法名为 GENETIC-2 （如线性特例）。

9.2.1 表 达

正如线性情况，我们选择二维结构表示运输问题的一个解（一个染色体）：矩阵 $V = (x_{ij})(1 \le i \le k, 1 \le j \le n)$ 。这时，每个 x_{ij} 是一实数。

9.2.2 初始化

初始化过程和线性情况相同（9.1.3 节）。和线性情况一样，它产生一个至多 $k+n-1$ 个非零元素的矩阵，使所有的约束都被满足。虽然其他初始化过程是可行的，这种方法将产生一个解，此解为一描述约束解空间凸边界的单纯形顶点。

9.2.3 评 价

在本情况下，我们必须求费用的最小值，矩阵表值的非线性函数。选择了一些函数（9.2.6 节），实算结果在 9.2.7 小节中进行了描述。

9.2.4 算 子

我们定义了两个遗传算子：变异和算术杂交。

- 变异

两种类型的变异算子被定义。第一个是 **mutation–1**, 与线性情况中使用的相同，并尽可能多地引入了零表值到矩阵里。第二个是 **mutation–2**，被修正以避免从一区间选择值时挑选零元素。除了使用了初始化(**initialization**)步骤的修正版对被挑选的子矩阵重新计算其内容以外，**mutation–2** 算子和 **mutation–1** 相同，

对 9.1.3 节中的描述做了如下的变化：下面的一行

set $val = \min(sour[i], dest[j])$

被下面的四行替换：

set val_1 = min(*sour*[*i*], *dest*[*j*])

if (*i* is the last available row) **or** (*j* is the last available column)
then $val = val_1$
else set *val* = random (real) number from ⟨ 0, val_1⟩

这种变化给出了实数，而不是整数和零，但是该过程必须进一步修正，因为它现在产生的矩阵违反约束。

例如，使用例 9.1 的矩阵，假定被选择的数序列为⟨ 3,6,12,8,10,1,2,4,9,11,7,5⟩对数 3（第一行，第三列）产生的第一个实数为 7.3（它在区间⟨ 0.0, min(*sour*[1], *dest*[3]) ⟩ = ⟨0.0, 15.0 ⟩里）。接着，数 6（第二行，第二列）产生的第二个随机实数为 12.1，等等，新初始化算法产生的其余的实数为：3.3, 5.0, 1.0, 3.0, 1.9, 1.7, 0.4, 0.3, 7.4, 0.5。 导出的矩阵为：

	5.0	15.0	15.0	10.0
15.0	3.0	1.9	7.3	1.7
25.0	0.5	12.1	7.4	5.0
5.0	0.4	1.0	0.3	3.3

只需将 1.1 加到元素 x_{11} 上，就满足约束。所以需要在 **mutation–2** 算法最后一行加上：

make necessary additions

这就完成了对 **initialization** 过程完整的修正。

• 杂交

杂交开始于两个亲体（矩阵 U 和 *V*），杂交算子将产生两个子代 *X* 和 *Y*，其中

$$X = c_1 \cdot U + c_2 \cdot V \text{ 及 } Y = c_1 \cdot V + c_2 \cdot U$$

（$c_1, c_2 \geq 0$ ； $c_1 + c_2 = 1$）。因为约束集是凸的，这种线性操作将确保：如果两个亲体是可行的，则两个子代也都是可行的。这是一个很重要的对线性情况的简化，另一个要求是维持矩阵中所有元素都是整数。

9.2.5 参 数

除了一组针对线性情况的控制参数外（群体规模、变异和杂交率、随机数起始种子等等），还需要一些其他的参数。这就是杂交比例，c_1 、c_2 和 m_1 — 用以确定 **mutation–1** 在应用的变异中的比例。

9.2.6 测 试

为显示建议的方法的实用性，我们选择了一个 7×7 的运输问题的简单例子（见表 9.2）及对各种目标函数的实算。

问题是最小化下列函数：

$$f(\mathbf{x}) = f(x_1, \ldots, x_{49})$$

并使之服从 14 个（13 个是独立的）等式约束：

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = 27$$
$$x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} = 28$$

$$x_{15}+x_{16}+x_{17}+x_{18}+x_{19}+x_{20}+x_{21}=25$$
$$x_{22}+x_{23}+x_{24}+x_{25}+x_{26}+x_{27}+x_{28}=20$$
$$x_{29}+x_{30}+x_{31}+x_{32}+x_{33}+x_{34}+x_{35}=20$$
$$x_{36}+x_{37}+x_{38}+x_{39}+x_{40}+x_{41}+x_{42}=20$$
$$x_{43}+x_{44}+x_{45}+x_{46}+x_{47}+x_{48}+x_{49}=20$$
$$x_1+x_8+x_{15}+x_{22}+x_{29}+x_{36}+x_{43}=20$$
$$x_2+x_9+x_{16}+x_{23}+x_{30}+x_{37}+x_{44}=20$$
$$x_3+x_{10}+x_{17}+x_{24}+x_{31}+x_{38}+x_{45}=20$$
$$x_4+x_{11}+x_{18}+x_{25}+x_{32}+x_{39}+x_{46}=23$$
$$x_5+x_{12}+x_{19}+x_{26}+x_{33}+x_{40}+x_{47}=26$$
$$x_6+x_{13}+x_{20}+x_{27}+x_{34}+x_{41}+x_{48}=25$$
$$x_7+x_{14}+x_{21}+x_{28}+x_{35}+x_{42}+x_{49}=26$$

表 9.2 7 × 7 运输问题

	20	20	20	23	26	25	26
27	x_1	x_2	x_3	x_3	x_5	x_6	x_7
28	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
25	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
20	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
20	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
20	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}
20	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}

非线性优化算法的行为显著地依赖于目标函数的格式。很明显，不同的求解技术可能产生十分不同的反应。

出于测试的目的，我们人为地将有潜力的目标函数分类成实用的运筹学(OR)问题中能看得明白的方式，这些实用问题主要在有关（比较好的）优化课本中看到并经常被作为其他的优化技术难度较大的测试特例。简单地说，它们可划分为如下类别：

- 实用函数

  典型的分段线性费用函数，它们经常出现在实用中，或者因为数据的限制及设备操作有不同的费用域。它们通常并不光滑，导数也必定不连续。它们经常会使通过近似将它们转变成可能的可微函数的梯度法难以求解。例如下面的函数 $A(x)$和$B(x)$。

- 有理函数

  这类函数是光滑的，通常是流量的简单的幂关系。它们可进一步分级成凸和凹函数。例如下面的函数 $C(x)$和 $D(x)$。

- 其他函数

  这些函数的典型特征是带有次优解的多谷或者多峰，对任何梯度法都存在困难。它们被构造成了优化算法严格的测试问题，我们推测它们不会经常出现在实用中。例如下面的函数 $E(x)$和 $F(x)$。

下面列出了每类目标函数在测试中用到的两个例子。它们都是解向量组分可分函数，且无交叉项。它们的图形的连续形式（已在 GAMS 系统中被修正）绘制于图 9.3 中。

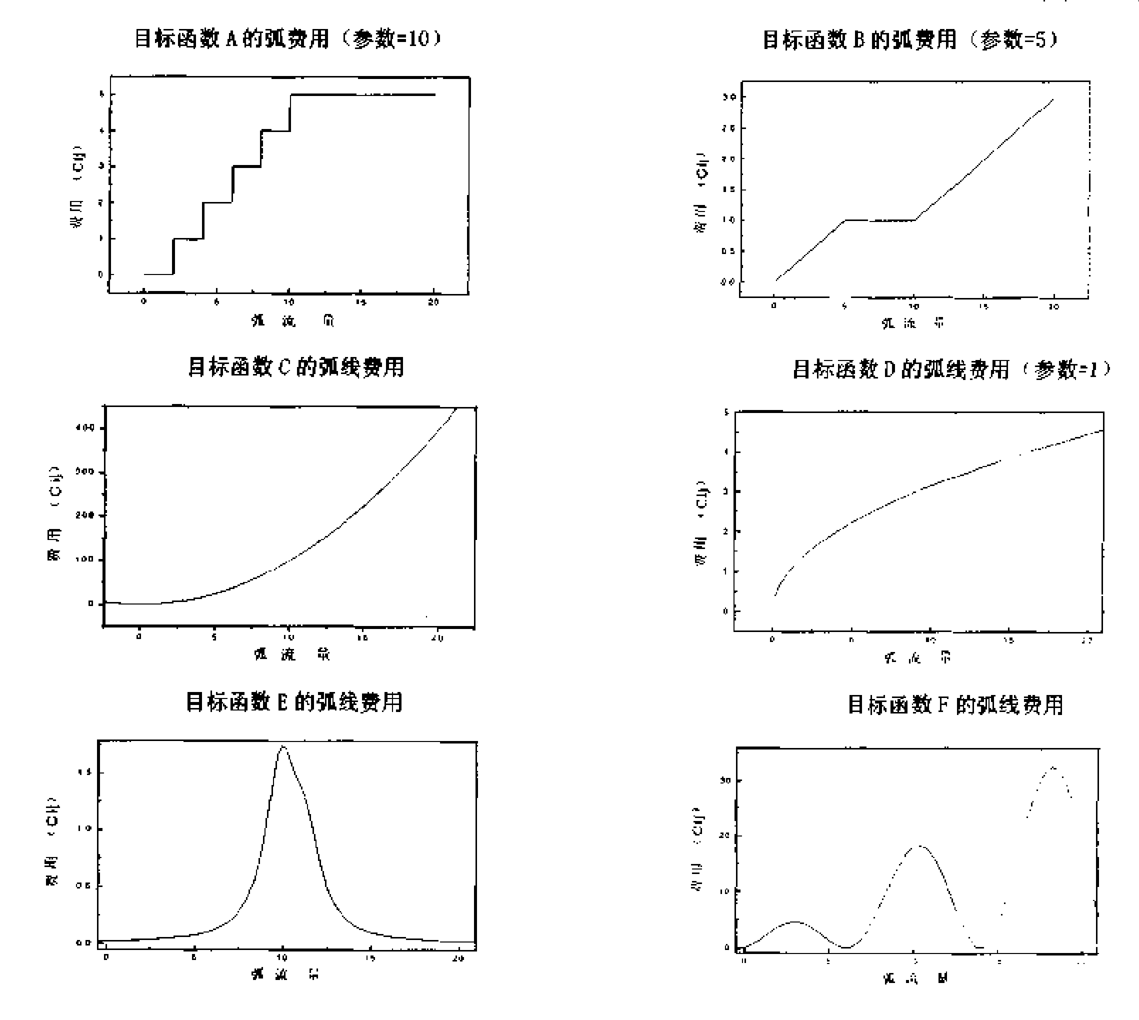


图 9.3 六个测试函数 A~F

- 函数 A

$$A(x)=\begin{cases}0 & 若\ 0<x\leq S\\ c_{ij} & 若\ S<x\leq 2S\\ 2c_{ij} & 若\ 2S<x\leq 3S\\ 3c_{ij} & 若\ 3S<x\leq 4S\\ 4c_{ij} & 若\ 4S<x\leq 5S\\ 5c_{ij} & 若\ 5S<x\end{cases}$$

其中 S 小于一个典型的 x 值。

- 函数 B

$$B(x)=\begin{cases}c_{ij}\dfrac{x}{S} & 若\ 0\leq x\leq S\\ c_{ij} & 若\ S<x\leq 2S\\ c_{ij}\left(1+\dfrac{x-2S}{S}\right) & 若\ 2S<x\end{cases}$$

其中 S 和典型的 x 值具有同样的阶。

- 函数 C

$$C(x)=c_{ij}x^2$$

- 函数 D

$$D(x)=c_{ij}\sqrt{x}$$

- 函数 E

$$E(x)=c_{ij}\left(\frac{1}{1+(x-2S)^2}+\frac{1}{1+(x-\frac{9}{4}S)^2}+\frac{1}{1+(x-\frac{7}{4}S)^2}\right)$$

其中 S 有典型的 x 值的阶。

- 函数 F

$$F(x)=c_{ij}x\left(\sin\left(x\frac{5\pi}{4S}\right)+1\right)$$

其中 S 和典型的 x 值有同样的阶。

运输问题的目标函数格式为：

$$\sum_{ij}f(x_{ij})$$

其中 $f(x)$为上述函数之一，参数 c_{ij} 是从参数矩阵中获得（见图 9.4），S 是从测试问题的属性中获得。

来源地数目:	7						
目的地数目:	7						
来源地流量:	27	28	25	20	20	20	20
目的地流量:	20	20	20	23	26	25	26

弧参数矩阵（来源地×目的地）：

0	21	50	62	93	77	1000
21	0	17	54	67	1000	48
50	17	0	60	98	67	25
62	54	60	0	27	1000	38
93	67	98	27	0	47	42
77	1000	67	1000	47	0	35
1000	48	25	38	42	35	0

图 9.4 例子问题的描述

为推导 S，需要估算 x 的典型值：可通过初步运行的方式估算非零 x_{ij} 的数目和大小。用这种方式可估算每个弧线的平均流量，找到 S 的值。对函数 A 使用 $S=2$，对 B、E 和 F，$S=5$。

注意，目标函数对每个弧线是相同的，所以，费用矩阵被用来指示弧线之间的变化。此矩阵提供了标度基本函数形状的 c_{ij}，这样就提供了可变性的“一个度”。

9.2.7 实算和结果

在测试对线性运输问题的 GENETIC–2 算法（见 9.1 节）里，我们与用标准算法找到的已知优化解的标准算法进行了比较。因此可以确定遗传算法在绝对意义上的效率。一

且移向非线性目标函数，最优解可能是未知的。因此，在将该算法和其他本身可能收敛到局部最优解的非线性求解方法比较时，我们减少了测试。

照例，在比较 GENETIC-2 算法时，用 GAMS 系统作为工业标准有效的方法的一个典型例子。该系统是梯度控制方法，帮我们找到一些能确定地说是难以解决或者不可能解决的问题。在这些特例中，需要对目标函数作一些修正，以便使所用的方法能至少得到一个近似解。

运输问题的目标函数形式是：

$$\sum_{ij} f(x_{ij})$$

其中 $f(x)$ 是六个选择函数之一，c_{ij} 参数从参数矩阵中获得，S 是从测试问题的属性中获得。S 可近似从平均非零弧线流量确定，该流量是用一些初步运行并确保流量发生在目标函数的感兴趣的部分来获得的。

在某种意义上，应该对每个弧使用完全的随机结构化目标函数。指定我们的目标是论证算法对各种问题的性能如何，那么问题就简化成：对一特定的函数形式，弧线之间的多少变化是所要求的。当对每个弧线的函数相同时，问题可能有费用相同的许多解，分析该算法，就可以整理所获得的信息量。

在我们的实算中，一个费用矩阵可用来提供弧线之间的变化。矩阵提供的 c_{ij} 主要是用来标度基本函数的形状，这样就提供了可变性的“一个度”。而不需更多的矩阵（提供可变性更多的度）。

对函数 C，E 和 F，GAMS 应用是直接的：使用固有的非线性函数。由于对目标函数的梯度估算的要求，GAMS 不能直接处理函数 A，B 和 D，对 A 和 B 的情况，表达在 GAMS 中不能公式化，而对 D 的情况（平方根函数），在度量靠近 0 的梯度时，遇到一些困难。因此，对 GAMS 的运行，我们对问题做了如下的修正：

- *函数 A*

  分离的弧切线函数被用来近似计算五个步骤中的每一个。参数 P_A，被用来控制合适的“紧度”。弧 $[i,j]$ 的费用为：

$$c_{ij}\cdot\left(\begin{array}{l}\arctan(P_A(x_{ij}-S))/\pi+\frac{1}{2}+\\ \arctan(P_A(x_{ij}-2S))/\pi+\frac{1}{2}+\\ \arctan(P_A(x_{ij}-3S))/\pi+\frac{1}{2}+\\ \arctan(P_A(x_{ij}-4S))/\pi+\frac{1}{2}+\\ \arctan(P_A(x_{ij}-5S))/\pi+\frac{1}{2}\end{array}\right)$$

- *函数 B*

  弧切线函数再度被使用，这次近似计算三个梯度中的每一个。参数 PB 用来控制合适的紧度，弧[i,j]的费用为：

$$c_{ij}\cdot\begin{pmatrix}(\frac{x_{ij}}{S})\cdot(\arctan(P_B x_{ij})/\pi+\frac{1}{2})+\\(1-\frac{x_{ij}}{S})\cdot(\arctan(P_B(x_{ij}-S))/\pi+\frac{1}{2})+\\(\frac{x_{ij}}{S}-2)\cdot(\arctan(P_B(x_{ij}-2S))/\pi+\frac{1}{2})\end{pmatrix}$$

- 函数 D

为避免靠近零的梯度问题，函数 D 变为：

$$D'(x)=D(x+\varepsilon)$$

如同线性的情况，对每个问题，GAMS 的多次运行是在修正参数的不同值上进行并挑选最好结果。找到的三个参数的最好值为：P_A 在 1 和 20 之间，P_B 很大（如 1000），ε（对函数 D）在 1 和 7 之间。最终结果值总是在使用未修正函数优化后计算得到的，而不是用修正函数。

主要的一组实算中，五个 10×10 运输矩阵被用在每个函数。它们是从一组独立的均匀分布的 c_{ij} 值和随机挑选总流量为 100 个单位的来源地和目的地向量来构造的。对遗传算法，每个函数矩阵的组合是使用不同随机数起始种子的 5 次运行中得到的值。问题运行 10000 代。

对函数 A，S 设定为 2，而对函数 B，E 和 F，S 值使用 5。

10×10 结点问题达到了 GAMS 学生版的极限(其中所允许的问题大小是受限制的)。从列出的用在 GAMS 系统上的一些例子问题的表中可以看出，运行一个 25×25 结点问题，用在 640K 内存的 AT 计算机的完整版本(其中问题的大小受可用内存和内部的局限)看来是可能的。注意一个 $N\times N$ 结点问题被 GAMS 表达成有 N^2 个变量，2N 个约束和一个非线性目标函数。很明显，较大的问题可以在较大的系统（特别是大型机）或者用特殊的解算器上明确表达。

不过，用太大的问题比较遗传系统和非线性规划型解算器却没有多大的价值。10×10 的运行结果表明 GAMS 以及可能的相似系统趋向于落到局部（非全局）最优解上。忽略花在评价目标函数和用在可作为时间度量的测试解的数目上的时间消耗，很明显的是标准的非线性规划技术总是比遗传系统更快地“完成”。这是因为，它们只是代表性的探求了在当前局部最优解区域之内的特殊路径。只有在局部最优解相对好时，它们才做得很好。

获得线性问题的经验后和对基于非线性问题的调节性运行，GENETIC-2 挑选了一套参数。群体规模固定为 40。变异率为 $p_m=20\%$ ，mutation-1 的比例为 50%，杂交率为 $p_c=5\%$。杂交比例为 $c_1=0.35$ 及 $c_2=0.65$。

比起经典的遗传算法来，好像所选的的变异率明显太高而杂交率太低。不过，我们的算子不同于经典的遗传算法，因为：（1）我们选择变异和杂交的亲体，即整体结构（相对于单个的位）来进行变异，（2）mutation-1 产生一个将亲体朝解空间表面“推”的后代，而杂交和 mutation-2 将后代朝解空间的中心“推”。

表 9.3　不同 p_c 和 p_m 值下的运行结果

函　数	p_c=0　p_m=25%	p_c=25%　p_m=0%	p_c=5%　p_m=20%
A	45.8	181.0	0.0
F	178.7	189.6	110.9

高变异率的使用还启示该算法几乎是一个随机搜索，但随机搜索算法（杂交率 0%）性能比起这里使用的调节算法十分差。为说明这一点，我们用不同参数值 p_m 和 p_c，并使用函数 A 和 F 的特定 7× 7 运输问题的一些典型结果列在前表 9.3 中。给出的值是用不同的种子在 10000 代的五次运行所达到的平均最小费用。

使用的 7 × 7 运输问题在图 9.4 中给出；GENETIC−2 算法对不同的 p_c 和 p_m 参数值下发现的解如图 9.5 所示。

p_c=0%，p_m=25%，函数 A，Cost=45.8

20.00	0.00	0.00	1.00	2.00	2.00	2.00
0.00	20.00	1.00	2.00	2.00	2.00	1.00
0.00	0.00	19.00	0.00	2.00	1.00	3.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	20.00

p_c=0%，p_m=25%，函数 F，Cost=179.7

15.00	6.00	0.00	6.00	0.00	0.00	0.00
0.00	14.00	0.00	14.00	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
0.00	0.00	0.00	3.00	6.00	0.00	6.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

p_c=25，p_m=0%，函数 A，Cost=181.2

18.25	0.00	0.11	1.81	3.30	3.53	0.00
0.00	18.21	3.94	3.05	0.00	1.97	0.82
1.75	1.79	13.24	1.46	1.46	1.48	3.83
0.00	0.00	1.91	14.87	1.18	0.35	1.69
0.00	0.00	0.72	0.97	18.10	0.21	0.00
0.00	0.00	0.02	0.71	1.96	17.30	0.00
0.00	0.00	0.05	0.14	0.00	0.16	19.65

p_c=25%，p_m=0%，函数 F，Cost=189.6

20.00	0.25	0.00	0.75	0.00	6.00	0.00
0.00	5.75	0.00	22.25	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
0.00	14.00	0.00	0.00	6.00	0.00	6.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	14.00	6.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

p_c=5%，p_m=20%，函数 A，Cost=0.0

19.87	0.00	0.68	1.80	1.33	1.80	1.51
0.08	20.00	1.00	1.90	1.48	1.61	1.92
0.00	0.00	18.32	1.89	1.08	1.78	1.93
0.05	0.00	0.00	17.09	1.91	0.96	0.00
0.00	0.00	0.00	0.00	19.92	0.00	0.08
0.00	0.00	0.00	0.00	0.00	18.60	1.40
0.00	0.00	0.00	0.31	0.28	0.25	19.16

p_c=5%，p_m=20%，函数 F，Cost=110.9

14.31	6.31	6.39	0.00	0.00	0.00	0.00
0.00	13.69	0.31	14.00	0.00	0.00	0.00
0.00	0.00	13.31	6.00	0.00	0.00	5.69
5.69	0.00	0.00	3.00	6.00	0.00	5.31
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	19.31	0.69
0.00	0.00	0.00	0.00	0.00	5.69	14.31

图 9.5　GENETIC−2 对不同的 p_c 和 p_m 值找到的解

GENETIC−2 系统是在 SUN SPARC station 1 工作站上运行的，而 GAMS 是在 Olivetti 386 上运行的。虽然对这两种机器之间的速度进行比较是困难的，但可注意到总的来说，对每次运行，GAMS 在遗传系统之前结束得好。一个例外是对情况 A（其中 GAMS 评价大量的弧切线函数），遗传算法花费了不足 15 分钟就完成，而 GAMS 平均两倍于此。对情况 A，B 和 D，额外的 GAMS 修正参数意味着为找到最好解，必须执行许多次，所以遗传系统总体上要更快些。

表 9.4　GAMS 和 GENETIC−2 的比较

函数	GAMS	GENETIC−2	误差%
A	281.0	202.0	−28.1%
B	180.8	163.0	−9.8%
C	4402.0	4556.2	+3.5%
D	408.4	391.1	−4.2%
E	145.1	79.2	−45.4%
F	1200.8	201.9	−83.2%

来源地数目:	10									
目的地数目:	10									
来源地流量:	8	8	2	26	12	1	6	18	18	1
目的地流量:	19	2	33	5	11	11	2	14	2	1

弧参数矩阵（来源地 × 目的地）

$$
\begin{pmatrix}
15 & 3 & 23 & 1 & 19 & 14 & 6 & 16 & 41 & 33 \\
13 & 17 & 30 & 36 & 20 & 17 & 26 & 19 & 3 & 33 \\
37 & 17 & 30 & 5 & 48 & 27 & 8 & 25 & 36 & 21 \\
13 & 13 & 31 & 7 & 35 & 11 & 20 & 41 & 34 & 3 \\
31 & 24 & 8 & 30 & 28 & 33 & 2 & 8 & 1 & 8 \\
32 & 36 & 12 & 9 & 18 & 1 & 44 & 49 & 11 & 11 \\
49 & 6 & 17 & 0 & 42 & 45 & 22 & 9 & 10 & 47 \\
2 & 21 & 18 & 40 & 47 & 27 & 27 & 40 & 19 & 42 \\
13 & 16 & 25 & 21 & 19 & 0 & 32 & 20 & 32 & 35 \\
23 & 42 & 2 & 0 & 9 & 30 & 5 & 29 & 31 & 29
\end{pmatrix}
$$

图 9.6 例子问题的描述

GENETIC–2（5 个种子的平均）和 GAMS 之间对求一单个的 10 × 10 问题的一个典型的比较在表 9.4 中展示；问题的描述如图 9.6 给出。

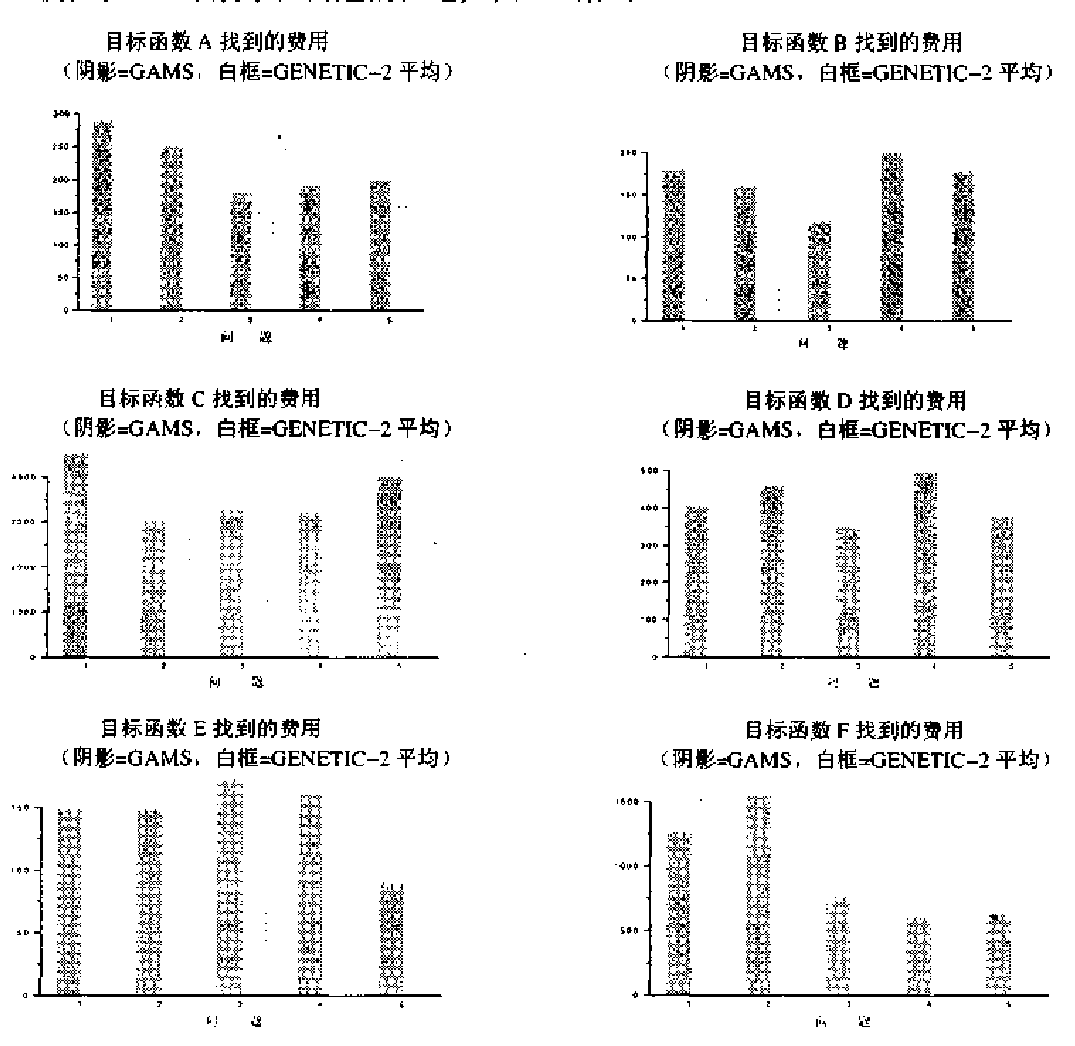


图 9.7 结果

图 9.7 显示了所有五个考虑的问题的结果。对一些“实际”问题，*A* 和 *B*，GENETIC–2

平均来说要好于GAMS ——对特例A为24.5%、对特例B为11.5%。对“有理函数”结果则是不同的。对特例C（平方函数），遗传系统性能要差7.5%，而对特例D（平方根函数），遗传系统平均只好于2.0%。对其他函数，E和F，遗传系统占主导地位：五个问题平均要比GAMS分别改进了33.0%和54.5%。

9.2.8 结论

我们的目标是研究多约束和非线性目标函数问题一类遗传算法的行为。挑选运输问题来研究是因为它提供了相对简单的凸可行集。这使对解的可行性进行维护更容易。然后我们就能检验目标函数对算法行为的单独影响。

结果证明，遗传方法在较难的问题上找到全局最优解是有效的，虽然GAMS在平滑单调的“有理”函数上也做得很好。梯度控制技术较适合这种情况。对函数C，GAMS找到更好解的速度要快于GENETIC–2。

对“实际”问题，梯度技术难以以更好费用在新区域的“角落周围寻找”。遗传类型算法，是采用一个更全局的方法，能迅速地转移到新区域，因此产生了更好的解。

对“其他”问题，虽然它们都是平滑的，但一个重要的结构特征是有意设计来造成梯度法难以求解。GENETIC–2对这些“实际”的特例，要优于GAMS。

比较GENOCOP （第7章）和GENETIC–2同样是有趣的（见表9.5）。总的来说，它们的结果十分相似。但是值得注意的是：矩阵法可以变成特殊的（运输）问题，而GENOCOP 是与问题无关的，不需任何难以解码的域知识。换句话说，可以预计对其他约束问题，GENOCOP也能执行得同样好，GENETIC–2却完全不能用。

表9.5 GENETIC–2对GENOCOP：对7×7问题的结果，运输费用函数A~F的费用矩阵在图7.3中给出

函数	GENETIC–2	GENOCOP	相差%
A	00.00	24.15	
B	203.81	205.60	0.87%
C	2564.23	2571.04	0.26%
D	480.16	480.16	0.00%
E	204.73	204.82	0.04%
F	110.94	119.61	7.24%

当比较所有三个系统（GAMS, GENOCOP, GENETIC–2）时，很重要的是突出它们中的两个，GAMS 和 GENOCOP, 是与问题无关的：它们有能力优化服从任何线性约束集的任何函数。第三个系统GENETIC–2是专门针对运输问题设计的：特定的约束被并入矩阵数据结构和特殊的“遗传”算子（更进一步的比较见第14章）。

GENETIC–2专门针对运输问题，但是一个重要的特征是：它能处理任何类型的费用函数（甚至不必是连续的）。同样可能对它进行修正以处理包括配置(allocation)和某些日程表(scheduling)的问题许多类似的运筹学问题。看起来，这是一个有前途的研究方向，它可以引导遗传技术解基于约束优化问题的矩阵。

第 10 章　货郎担问题

在下一章，我们将推出适合特定应用的几个演化程序的例子（作图、分割、调度 —— graph drawing, partitioning, scheduling）。货郎担问题（Traveling Salesman Problem, TSP）只是其中一个；但我们把它作为一个特殊的问题处理——所有问题的样本——并在本章首先单独讨论它，是什么理由呢？

有许多原因。首先，TSP 概念上很简单：满处跑的货郎必须在其活动领域对每个城市的访问次数精确地为一次，并返回到起始点。给定所有城市之间的旅行费用，应该如何计划其行程使整个旅行的费用最小？TSP 的搜索空间是 n 个城市的排列集合。任何单个的 n 个城市的排列产生一个解（对 n 个城市的完全旅行）。优化解是产生旅行最小费用的排列。搜索空间的大小是 $n!$。

TSP 是一个相当老的问题：早在 1759 年，Euler 就研究过（虽然当时不叫这个名字），他的研究兴趣是骑士的旅行问题。一个正确的解是：在访问过程中，到达象棋盘中的 64 个方块中的每一处精确地为一次。

术语“货郎”第一次使用是在 1932 年出版的一本德国书《货郎应该如何做及做什么以便在生意上获得成功》，是由一位老练的推销员写的（见参考文献[236]）。尽管不是此书的主题，TSP 和调度在此书的最后一章进行了讨论。

TSP 是由 RAND 公司（RAND Corporation）于 1948 年引入的。该公司的声誉使 TSP 成为一个知名且流行的问题。TSP 在那时变得流行是因为新的课题：线性规划和组合问题。

货郎担问题已被证实是一个 *NP* 难解问题[134]。它出现在许多应用中，且“城市”的数量可能十分大 — 正如[207]中所描述的：

> “[246]中描述的电路板钻孔应用有 17000 个城市，[44]中描述的 X 射线晶体学例子中有 14000 个城市，报告的 VLSI 构造中有一百二十万个城市之多[227]。而且，如果一个人能在一台 PC 机上用百分之几秒（为某个问题）能获得的一个解，而在一台几百万美元的计算机上（为某个问题）求最优解消耗 5 个小时，则可能不是在成本上合算的。这就是为什么需要启发性规则的原因。”

在过去 10 年中，出现了一些逼近最优解的算法：最近邻居(nearest neighbor)、贪婪算法(greedy algorithm)、最近插入(nearest insertion)、最远插入(farthest insertion)、双最小生成树(double minimum spanning tree)、带解法(strip)、空间充填曲线(space–filling curve)、及由 Karp，Litke，Christofides 等开发的算法[207]。（其中的一些算法假定城市对应于一个标准度量平面里的点）。另一类算法（2-opt，3-opt，Lin–Kernighan）则是针对局部优化：用局部扰动对一次旅行进行改进。TSP 也成为遗传算法界的一个目标：发表了许多基于遗传的算法研究报告，如[131], [160], [168], [170], [206], [241], [288], [299], [353], [370],

[375], [389]及[402]。这些算法旨在通过维持潜在解的群体来产生近似最优解，这些解将以偏向适应个体的选择模式经历一些一元和二元的转换（“变异”和“杂交”）。比较这些方法是很有趣的，应予特别注意的是使用的表达和遗传算子——这就是我们在本章中打算这样做的原因。换句话说，我们将跟踪 TSP 的演化程序的进化过程。

为勾勒出 TSP 的一些重要特征，让我们首先考虑满足 CNF 的问题。合取范式(conjunctive normal form，CNF)的逻辑表达是一个被布尔算子“∧”分开的子句序列；一个子句是一个被布尔算子 ∨ 分开的文字序列；一个文字是一个逻辑变量或其非值；一个逻辑变量是可以赋予值为 TRUE 或者 FALSE （1 或 0）的变量。

例如在 CNF 中下面的逻辑表达：

$$(a \vee \bar{b} \vee c) \wedge (b \vee c \vee d \vee \bar{e}) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{c} \vee \bar{e})$$

其中 a，b，c，d 和 e 为逻辑变量；$\bar{a}$ 表示变量 a 的非值 （当且仅当 a 的值为 FALSE，$\bar{a}$ 的值为 TRUE）。

问题是确定是否存在表达中变量的一个真假值赋值，使整个表达被评价为 TRUE。例如，上述 CNF 逻辑表达有几个确实的真假值赋值，对 a = TRUE 及 c = TRUE 的任何其他赋值，整个表达评价值为 TRUE。

如果我们试图应用遗传算法到满足 CNF 的可满足性问题，我们注意到，很难想象其他更适合问题的表达：一个定长的二进制向量（向量的长度对应于变量数）即满足这样的任务。而且，在位之间没有相关性：任何变化将产生合法（有意义的）的向量。这样，我们可以应用杂交和变异，而无需任何解码器或者修补算法。但是，评价函数的挑选是最难的工作。注意所有的逻辑表达评价值为 TRUE 或者 FALSE，如果一个特定的真假值赋值评价整个表达为 TRUE，那么对问题的解就找到了。要点是在搜索一个解的过程中，群体中的所有的染色体（向量）将被评价为 FALSE（除非一个解已经发现），所以不可能区分“好”和“差”解。简而言之，可满足性 CNF 问题有自然的表达和算子，而没有自然的评价函数。有关对适当的评价函数的选择问题的更进一步的讨论，读者可以参考[90]。

另一方面，TSP 有相当容易且自然的评价函数：对任何潜在解（城市的排列），我们可以引用所有城市之间的距离表，而且（经过 n–1 次另外的操作）我们获得旅行的整个长度。这样，在一旅行群体中，我们可以很容易地比较它们中的任何两个。但是，一个旅行表达的挑选和使用的算子的挑选却还远不清楚。

在单独一章里处理 TSP 的另一个理由是相似的技术被广泛用在各种其他的排序问题中，像调度和分割问题。其中的一些将在下一章讨论。

在遗传算法界有一个共识：旅行的二进制表达对 TSP 不是最适合的。很难明白这是为什么：毕竟，我们感兴趣的是城市的最好排列，即

$$(i_1, i_2, \ldots, i_n)$$

其中 $(i_1, i_2, \ldots, i_n)$ 是 $\{1,2,\ldots,n\}$的一个排列。这些城市的二进制代码没有任何优点。恰恰相反：二进制表达要求特殊的修补算法，因为单个位的变化可能引起一个非法的旅行。

正如[402]中的评述：

> “不幸的是，没有一个实用的方式来对TSP二进制串进行解码，该二进制串没有次序的依赖关系或者不能用一个有意义的样式应用算子。简单的城市交叉串产生重复和删除。因此，为解决这样的问题，必须对一些标准遗传杂交进行改变。理想的重组算子将以一种非破坏性的、有意义的方式重组亲体结构的重要信息。”

值得注意的是，Lidd的近期文章[241]描述了一种运用二进制表达和经典的算子（杂交和变异）的针对TSP的GA方法。非法旅行的评价是基于由贪婪算法(greedy algorithm)产生的完全旅行（不必要是合法的）。此报告的结果有令人惊讶的高质量，但考虑的最大测试例子只是由100个城市组成。

在过去几年，与TSP有关的有三种向量表达：邻接、普通及路径(adjacency, ordinal, path)表达。每种表达有其自身的“遗传”算子，我们将依次对它们进行讨论。因为提供某类变异算子是相对容易的，算子引入一个小变化到旅行中。我们将集中讨论杂交算子。在所有的三个表达中，一次旅行被描述为一个城市表。在下面的讨论中，我们使用从1到9有9个编号城市的TSP作为一个共同的例子。

1．邻接表达

邻接表达表示有 n 个城市表的旅行。当且仅当旅行是从城市 i 通向城市 j 时，城市 j 列在位置 i 上。例如向量：

$$(2\ 4\ 8\ 3\ 9\ 7\ 1\ 5\ 6)$$

表示下面的旅行：

$$1-2-4-3-8-5-9-6-7$$

每个旅行只有一个邻接表表达；但是，一些邻接表可能表达非法的旅行，如，

$$(2\ 4\ 8\ 1\ 9\ 3\ 5\ 7\ 6)$$

它将导致：　$1-2-4-1$

即部分旅行是一个过早的循环。

邻接表达不支持经典的杂交算子。修补算法是必要的。邻接表达有三种杂交算子被定义并被研究：交互边(alternating edges)杂交、子旅行块(subtour chunks)杂交及启发式杂交[168]。

- 交互边杂交通过从第一个亲体中随机选择一个边(edge)来建立一个后代，然后从第二个亲体中选择一个适当的边，等等——该算子通过从交互的亲体中挑择边来扩展旅行。如果新的边（从一个亲体中）引入一个循环到当前（仍然是部分的）旅行中，算子将从不引入循环的剩余的边中随机选择替代边。例如来自于两个亲体：

  $$p_1=(2\ 3\ 8\ 7\ 9\ 1\ 4\ 5\ 6)$$
  $$p_2=(7\ 5\ 1\ 6\ 9\ 2\ 8\ 4\ 3)$$

  中的第一个后代可能是

  $$o_1=(2\ 5\ 8\ 7\ 9\ 1\ 6\ 4\ 3)$$

其中过程开始于来自亲体 p_1 的边(1, 2)，在替换边的过程中引入的仅有的一条随机边是(7, 6)，而不是(7, 8)，因为它将引入一个过早的循环。

- 子旅行块杂交从一个亲体中挑选一随机长度的子旅行，然后从另一个亲体中挑选另一个随机长度的子旅行来构造一个后代——此算子通过替换亲体中挑选边来扩展旅行。如果一些边（来自于其中一个亲体）引入一个循环到当前（仍然是部分的）旅行，算子从剩余边选择一不引起循环的随机边。
- 启发式杂交通过挑选一随机城市作为后代旅行的开始点来建立一个后代，然后比较两条边（都来自于亲体），保留此城市并选择较好（较短的）边。被选择的边的另一端的城市作为选择离开该城市两条边的较短边的开始点。如果在某个阶段，一条新的边将引入一循环到部分旅行中，那么，旅行通过从剩余不引起循环的边中选择的随机边来扩展。

  在[206]中，作者通过改变两个法则修正了上述启发式杂交：（1）如果较短的边（来自于一个亲体）引入了一个循环到一个后代旅行中，则检查另一条（较长的）边。如果较长的边不引入一个循环，则接受它，否则（2）从 q 个随机选择边中选择最短的边（q 为此方法的一个参数）。

  该算子的效果是联结亲体旅行中短的子路径在一起。但是，它可能生成不应有的边交叉——这就是为什么启发式杂交对旅行的局部微调是不合适的原因。Suh 和 Gucht[375]引入了另一个启发式算子（基于二优算法[245]）适合于局部调节。该算子随机地选择两条边，$(i\ j)$ 和 $(k\ m)$，并检查是否

  $$dist(i,j)+dist(k,m)>dist(i,m)+dist(k,j)$$

  其中 $dist(a,b)$为城市 a 和 b 之间的给定距离。如果属于这种情况，旅行中的边 $(i\ j)$ 和$(k\ m)$ 被边 $(i\ m)$ 和 $(k\ j)$ 替换。

邻接表达的一个优点是它允许进行第 3 章中讨论过的类似的模式分析(shemata analysis)，其中二进制串被考虑。它对应于自然基因块的模式，即边；如模式

$$(*\ *\ *\ 3\ *\ 7\ *\ *\ *)$$

表示有边 (4 3) 和 (6 7) 的所有旅行集合。但是这种表达的主要缺点是对所有算子都是相对不良的结果。选择边杂交常会破坏好的旅行，因为从两个亲体中交互边自身的操作。子旅行块杂交比交互边杂交执行得更好，原因是破坏率较低。然而它的执行效率仍然十分低，当然，启发式杂交是这里最好的算子。理由是前两个杂交是盲目的，即它们并不涉及实际的边的长度。而启发式杂交从两个可能的边中选择较好的边——这就是它比其他两个执行得好的原因。但是，启发式杂交并不是杰出的：在报道的三个[168]实算中有 50、100 和 200 个城市，系统大约分别在 15 000、20 000 和 25 000 代里找到的旅行在最优解的比率在 25%、16%和 27%以内。

2. 普通表达

普通表达以一个有 n 个城市的表来表示一个旅行；表中的第 i 个元素为区间 1 到 $n-i+1$ 范围内的一个数。普通表达的思想如下。存在城市的次序表 C，可用于普通表达里表的

参照点。假定，有这样一个简单的次序表（参照点）：

$C=(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$。

一个旅行

1—2—4—3—8—5—9—6—7

被表示为参考表 l,

$l=(1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$

并应该作如下解释：

- 表 l 中的第一个数是 1，所以从表 C 中取出第一个城市作为旅行的第一个城市（城市号 1），并从 C 将它移走。部分旅行为 ：

  1

- 表 l 中的下一个数也是 1，所以从当前的表 C 中取出第一个城市作为旅行的下一个城市（城市号 2），并从 C 中移走它。部分旅行为：

  1—2

- 表 l 中的下一个数是 2，所以从当前的表 C 中取出第二个城市作为旅行中的下一个城市（城市号 4），并从 C 中移走它。部分旅行为：

  1—2—4

- 表 l 中的下一个数是 1，所以从当前表 C 中取出第一个城市作为旅行中的下一个城市（城市号 3），并从 C 中移走它。部分旅行为：

  1—2—4—3

- 表 l 中的下一个数是 4，所以从当前表 C 中取出第四个城市作为旅行中的下一个城市（城市号 8），并从 C 中移走它。部分旅行为：

  1—2—4—3—8

- 表 l 中的下一个数又是 1，所以从当前表 C 中取出第一个城市作为旅行中的下一个城市（城市号 5），并从 C 中移走它。部分旅行为：

  1—2—4—3—8—5

- 表 l 中的下一个数是 3，所以从当前表 C 中取出第三个城市作为旅行中的下一个城市（城市号 9），并从 C 中移走它。部分旅行为：

  1—2—4—3—8—5—9

- 表 l 中的下一个数是 1，所以从当前表 C 中取出第一个城市作为旅行中的下一个城市（城市号 6），并从 C 中移走它。部分旅行为：

  1—2—4—3—8—5—9—6

- 表 l 中的最后应该数是 1，所以从当前表 C 中取出第一个城市作为旅行的下一个城市（城市号 7，最后一个可用的城市），并从 C 中移走它，最终的旅行为：

  1—2—4—3—8—5—9—6—7

普通表达的主要优点是能进行经典的杂交。普通表达中的任何两个旅行在某个位置后切开并一起进行交叉，产生的两个后代中的每一个都是合法的旅行。例如，两个亲体：

$p_1=(1\ 1\ 2\ 1\ |\ 4\ 1\ 3\ 1\ 1)$

$p_2=(5\ 1\ 5\ 5\ |\ 5\ 3\ 3\ 2\ 1)$

它对应于旅行

1—2—4—3—8—5—9—6—7

5—1—7—8—9—4—6—3—2

杂交点以“|”标记，将生成下面的后代：

o_1=(1 1 2 1 5 3 3 2 1) ，o_2=(5 1 5 5 4 1 3 1 1)

这些后代对应于

1—2—4—3—9—7—8—6—5

5—1—7—8—6—2—9—3—4

可很容易地看出，杂交点左边的部分旅行没有变化，而杂交点右边的部分旅行可以以一种随机的方式打破。较差的实算结果表明[168]这种表达和经典杂交一起用对 TSP 并不适合。

3. *路径表达*

路径表达可能是对一个旅行最自然的表达。例如，一个旅行

5—1—7—8—9—4—6—2—3

可以简单地被表达成

(5 1 7 8 9 4 6 2 3)。

直至最近为路径表达定义了三种杂交：*部分映射*(partially-mapped, PMX)杂交、*次序*(order, OX)杂交及*循环*(cycle, CX)杂交。现在我们依次讨论它们。

- **PMX**——由 Goldberg 和 Lingle[160]建议的 — 通过从一个亲体中挑选一个旅行的子序列并尽可能多地保存另一个亲体的城市次序和位置来建立一个后代。一个旅行的子序列是通过挑选两个随机的切割点作为交换操作的边界来选择的。例如，两个亲体（切割点以“|”标记）

  p_1=(1 2 3 | 4 5 6 7 | 8 9)

  p_2=(4 5 2 | 1 8 7 6 | 9 3)

  按下面的方式产生后代。首先，切割点之间的片断被交换（符号“x”可以被解释成“现在未知”）：

  o_1=(x x x | 1 8 7 6 | x x)

  o_2=(x x x | 4 5 6 7 | x x)

  交换还定义了一系列的映射：

  $1\leftrightarrow 4$、$8\leftrightarrow 5$、$7\leftrightarrow 6$、$6\leftrightarrow 7$

  然后，我们可以（从原始的亲体中）填加更远的城市，但要做到不发生冲突：

  o_1=(x 2 3 | 1 8 7 6 | x 9)

  o_2=(x x 2 | 4 5 6 7 | 9 3)

  最后，后代 o_1 的第一个 x（它应该是 1，但有冲突）被 4 替换，因为有映射 $1\leftrightarrow 4$。相似地后代 o_1 的第二个 x 被 5 替换，后代 o_2 的 x 和 x 被 1 和 8 替换。这样后代为：

  o_1=(4 2 3 | 1 8 7 6 | 5 9)

  o_2=(1 8 2 | 4 5 6 7 | 9 3)

当使用适当的再生方案，PMX 杂交同时开拓了值和次序的重要相似性[160]。

- **OX**——由 Davis[71]提出 — 通过从一个亲体中挑选一个子序列旅行并保存另一个亲体的城市相对次序来构造后代。例如，两个亲体（切割点以“|”标记）

  $p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$

  $p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$

  将按照下面的方式产生后代。首先，切割点之间的片断被拷贝到后代里：

  $o_1 = (x\ x\ x \mid 4\ 5\ 6\ 7 \mid x\ x)$

  $o_2 = (x\ x\ x \mid 1\ 8\ 7\ 6 \mid x\ x)$

  下一步，从一个亲体的第二个切割点开始，将另一个亲体的城市以同样的次序被拷贝，略去已存在的符号，接到串的尾部，再从串的第一个位置继续。第二个亲体里的城市序列（自第二个切割点）为

  9—3—4—5—2—1—8—7—6;

  移走已在第一个后代里的城市 4、5、6 和 7 后，我们得到

  9—3—2—1—8

  该序列被放置在第一个后代里（从第二个切割点开始）：

  $o_1 = (2\ 1\ 8 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$

  相似地，我们可以得到另一个后代：

  $o_2 = (3\ 4\ 5 \mid 1\ 8\ 7\ 6 \mid 9\ 2)$

  OX 杂交开拓了路径表达的一个特性，即城市的次序（不是它们的位置）是重要的，即两个旅行

  9—3—4—5—2—1—8—7—6

  4—5—2—1—8—7—6—9—3

  实际上是相同的。

- **CX**——由 Oliver[299]提出 — 以这样的方式构造后代：每个城市（及其位置）均来自亲体中的一个。我们用下面的例子解释循环杂交的机制。两个亲体：

  $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$

  $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

通过抽取第一个亲体的第一个城市产生第一个后代：

$o_1 = (1\ x\ x\ x\ x\ x\ x\ x\ x)$

因为后代中的每个城市取自其亲体之一（从同样的位置），我们现在没有其他的挑选：被考虑的下一个城市必须是城市 4，由于此城市在亲体 p_2 中正好位于被选择城市 1“之下”。在 p_1 里，该城市在位置“4”上，所以有：

$o_1 = (1\ x\ x\ 4\ x\ x\ x\ x\ x)$

接着是城市 8，因为该城市亲体在 p_2 中正好位于被选择城市 4“之下”，于是有

$o_1 = (1\ x\ x\ 4\ x\ x\ x\ 8\ x)$

遵循这样的法则，第一个后代中应包含的下面的城市为 3 和 2。注意，城市 2 的选择要求选择城市 1，它已在表中 — 这样我们完成了一个循环：

$o_1 = (1\ 2\ 3\ 4\ x\ x\ x\ 8\ x)$

从另一个亲体填入的剩余的城市是：

$o_1 = (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)$

相似地， $o_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9)$

CX保存亲体序列中的元素的绝对位置。

还可能定义其他路径表达算子。例如Syswerda[383]定义了次序杂交算子的两个修正版本（但是该工作是和下一章的调度问题相关的，我们将在下一章讨论调度问题）。第一个修正（称为基于次序的杂交）随机地选择一个向量里的几个位置，一个亲体中被选择位置的城市的次序被强加到另一个亲体的对应城市上。例如，考虑两个亲体

$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$

$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

假定被选择的位置是3、4、6和9；亲体p_2的这些位置的城市次序将被强加到亲体p_1上。在亲体p_2中这些位置的城市（按给定的次序）为2、8、6和5。在亲体p_1中，这些城市位于位置2、5、6和8。后代中的这些位置元素重新安排次序以和p_2同样的元素次序相配（次序为2—8—6—5）。第一个后代为除了位置2、5、6和8以外p_1所有位置的一个拷贝：

$o_1 = (1\ x\ 3\ 4\ x\ x\ 7\ x\ 9)$

所有其他元素按照亲体p_2给出的次序即2、8、6、5被填入，所以最终：

$o_1 = (1\ 2\ 3\ 4\ 8\ 6\ 7\ 5\ 9)$

相似地，我们可以构造第二个后代：

$o_2 = (3\ 1\ 2\ 8\ 7\ 4\ 6\ 9\ 5)$

第二个修正（称为基于位置的杂交）和原来的次序杂交更相似。唯一的不同不是选择被拷贝城市的子序列，而在此场合下几个城市是随机地被选择的。

值得注意的是这两个算子（基于次序的杂交和基于位置的杂交）在某种意义上是彼此等价的。基于次序的杂交用一些选择的位置作为杂交点，基于位置的杂交用相应的位置作为杂交点，它们总是产生相同的结果。这就意味着如果杂交点的平均数是$m/2$（m为总的城市数），这两个算子将给出相同的执行效果。但是，如果杂交点平均数，譬如说为$m/10$，那么两个算子将展示不同的特征。有关这些算子更多的信息及对它们的理论和实算的比较，读者可以参考[154], [131], [299], [370]和[382]。

在概述过去一些年出现的不同的重新排序算子时，我们也应该描述一下一些倒置算子。简单的倒置[188]沿着染色体长度选择两个点，在这些点被切割，这些点之间的子串被颠倒。例如一个染色体：

(1 2|3 4 5 6|7 8 9)

以“|”标记两个切割点，被变成

(1 2|6 5 4 3|7 8 9)

这样，简单的倒置保证产生的后代是合法的旅行；一些理论研究[188]表明此算子在发现好串次序时是有用的。有报导[402]说在50个城市的TSP里，一个带有倒置的系统比一个带有“交叉和校正”算子的系统执行得更出色。但是，杂交切割点的数目的增加将降低

系统的执行效率。同样，倒置（类似变异）是一个一元算子，它只是补充重组算子——此算子本身不能重组信息。倒置算子的几个版本已被研究[154]。Holland[188]提供了一个模式定理的修正包含了其效果。

这里，我们应该同样陈述一下最近用演化策略解决 TSP 的一些尝试[178, 352]。这些尝试之一[178]，实算了四个不同的变异算子（变异在演化策略中仍是基本算子——见第 8 章）：

- 倒置——如上所示；
- 插入——选择一个城市并将它插入到一随机位置；
- 移位——选择一子旅行并将它插入到一随机位置；
- 互换——交换两个城市。

还有一个启发式算子的版本被使用。在这种修正里，几个亲体为产生后代作贡献。在后代旅行随机选择第一个城市后，该城市所有的左边的和右边的邻居（来自所有亲体）被检测。选择产生城市最短距离的城市。该过程继续下去直到旅行完成。

演化策略的另一个应用[352]产生一个有 n 个数的浮点向量（n 对应于城市数）。演化策略为所有的连续问题所应用。诀窍是编码。向量的组分被排序，而且它们的次序确定了旅行。例如，向量

$$v = (2.34, -1.09, 1.91, 0.87, -0.12, 0.99, 2.13, 1.23, 0.55)$$

对应于旅行

$$2—5—9—4—6—8—3—7—1$$

因为最小数-1.09 是向量 v 的第二个成员，第二最小数-0.12 是向量 v 的第五成员，等等。

到目前为止讨论的多数算子都是考虑的城市（即它们的位置和次序）而不是边——城市之间的连接。重要的可能不是旅行中一个城市的特殊位置，而是该城市和其他城市之间的连接。正如 Homaifar 和 Guan[194]所评述的：

> “仔细考虑一下问题，我们可以争辩说 TSP 的基本基因块是边而不是城市的位置表达。一个给定位置而没有邻接或者周围信息的城市或者短的途径对构筑好的旅行是没有什么意义的。但是，很难说清是否在位置 2 注入城市 a 比将其注入到位置 5 好。虽然这只是一种极端的情况，根本的假设是一个好的算子应该从亲体中尽可能多的抽取边的信息。这种假设可以从 Oliver 的文章[299]中的实算结果获得部分解释，其中 OX 比 PMX 好 11%，比 CX 好 15%。”

Grefenstette[170]开发了一类启发式算子来强调处的处理。其工作的主线是：

(1) 随机选择一个城市作为后代当前城市 c；

(2) 选择四条边（每个亲体两条）附加到当前城市 c 上；

(3) 定义一个基于费用的被选择边概率分布。连接先前访问过的城市边的概率为 0；

(4) 选择一条边，如果至少一条边有非零概率，则选择是基于上述分布的；否则选择是随机的（从未访问城市中）；

(5) 被选择边的“另一端”的城市成为当前城市 c；

(6) 如果旅行完成，停止；否则转到步骤(2)。

但是，正如[170]中所报告的，这样的算子从亲体中转移的边在 60%左右——这就意味着 40%的边是随机选择的。

Whitley、Starweather 和 Fuquay[402]开发了一个新的杂交算子：边重组杂交（edge recombination crossover, ER），它能从亲体中迁移超过 95%的边到一单个后代中。ER 算子探寻了旅行中边的信息。譬如说旅行

(3 1 2 8 7 4 6 9 5)

边为(3 1)、(1 2)、(2 8)、(8 7)、(7 4)、(4 6)、(6 9)、(9 5)和 (5 3)。 毕竟，是边而不是城市携带 TSP 的距离值。求最小的目标函数是其总的边数构造一个合法旅行。旅行中城市的位置不是很重要的：因为旅行是环形的。同样，边的方向也不重要：边(3 1) 和 (1 3)两个都只是表示城市 1 和 3 是直接相连的。

ER 杂交的总体思想是后代应该从两个亲体呈现的边来专门地构造。这是由两个亲体旅行产生的边表的帮助来完成的。对每个城市 c，边表提供至少在一个亲体中所有与城市 c 相连的城市，很明显，对每个城市 c，表中至少有两个，至多有四个城市。例如，对两个亲体

$$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$
$$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$$

边表为

city 1: edges to other cities: 9 2 4（城市 1：边通向其他城市的有 9 2 4）
city 2: edges to other cities: 1 3 8
city 3: edges to other cities: 2 4 9 5
city 4: edges to other cities: 3 5 1
city 5: edges to other cities: 4 6 3
city 6: edges to other cities: 5 7 9
city 7: edges to other cities: 6 8
city 8: edges to other cities: 7 9 2
city 9: edges to other cities: 8 1 6 3

后代的构造开始于从其中一个亲体中选择一个初始的城市。在[402]中，作者从初始城市中选择了一个（譬如上述例子中的 1 或 4）。在边表中有最小边数的城市被选择。如果这些数相同，则作一个随机挑选。这样的选择增加了从亲体选择所有的边以完成一个旅行的机会。用随机选择，使边失败，即剩下一个城市而没有连续的边的机会很高。假定我们已选择城市 1。该城市直接与三个其他城市相连：9、2 和 4。从这三个城市中选择下一个城市。在我们的例子中，城市 4 和 2 有三条边，城市 9 有四条。在城市 4 和 2 之间需作一个随机选择；假定城市 4 被选择。再往下构造的下一个城市的候选者是城市 3 和 5，因为它们直接与最后的城市 4 相连。接着城市 5 被选择，因为它只有三条边，而相对的城市 3 有四条边。到目前为止，后代的形状如下：

$(1\ 4\ 5\ x\ x\ x\ x\ x\ x)$

继续这一过程，直至终结，后代是

(1 4 5 6 7 8 2 3 9)

它是完全由取自两个亲体的边组成。从一系列的实算[402]，边失败发生的比率很低（1% ～ 1.5%）。

ER 算子测试[402]了有 30、50 和 75 个城市的三个 TSP — 在所有的情况下，返回的解都好于先前“已知最好”的序列。

两年后，边重组杂交得到更进一步的增强[370]。其思想是“公共子序列”不保存在 ER 杂交里。例如，如果边表包含三条边

city 4： edges to other cities： 3 5 1

这些边中的一条本身是重复的。参考先前的例子，它是边 (4 5)。该边存在于两个亲体中。不过，列出其他边，即(4 3) 和 (4 1)，它只存在于一个亲体中。建议的解法[370]用存储“标志”城市来修正边表：

city 4： edges to other cities： 3 –5 1;

符号“–”简单地指标志的城市 5 被列两次。先前的例子中的两个亲体

$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$

$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

增强的边表为：

city 1： edges to other cities： 9 –2 4

city 2： edges to other cities： –1 3 8

city 3： edges to other cities： 2 4 9 5

city 4： edges to other cities： 3 –5 1

city 5： edges to other cities： –4 6 3

city 6： edges to other cities： 5 –7 9

city 7： edges to other cities： –6 –8

city 8： edges to other cities： –7 9 2

city 9： edges to other cities： 8 1 6 3

构造新后代的算法给标志的表值以优先权：重要的是只有在三条边被列出的情况，而其他两种情况或者没有标志城市，或者两个城市都标志。这种增强（加上一个修正用以必须进行随机边选择时作更好的挑选）更进一步改进了系统的性能[370]。

边重组算子清楚地表明路径表达可能太差而不能表达一个旅行的重要特性 — 这就是为什么它被边表补充的原因。有其他更适合货郎担问题的表达吗？对此，我们不能给出肯定的“是”。但是确实值得去实验其他表达，如可能是非向量表达。

在最近两年，至少有三个独立的尝试：用矩阵表达作为染色体以构造一个演化程序。这是由 Fox 和 McMahon[131]、Seniw[353]及 Homaifar 和 Guan[194]做的，我们将简要地对它们依次讨论。

Fox 和 McMahon[131]将一个旅行表达成一个优先的二进制矩阵 M。当且仅当在旅行中，城市 i 发生在城市 j 之前，行 i 和列 j 的矩阵元素 m_{ij} 包含一个 1。例如，一个旅行

(3 1 2 8 7 4 6 9 5)

以矩阵形式表达如图 10.1 所示。

在该表达中，$n \times n$ 矩阵 M 表示一个旅行（城市整个次序）有下面性质：

- 1 的数目精确地为 $\frac{n(n-1)}{2}$，
- 对所有 $1 \le i \le n, m_{ii} = 0$。
- 如果 $m_{ij} = 1$ 且 $m_{jk} = 1$，那么 $m_{ik} = 1$。

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1
4	0	0	0	0	1	1	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1
7	0	0	0	1	1	1	0	0	1
8	0	0	0	1	1	1	1	0	1
9	0	0	0	0	1	0	0	0	0

图 10.1　一个旅行的矩阵表达

如果矩阵中 1 的数目小于 $\frac{n(n-1)}{2}$，且满足其他两个要求，则城市为部分次序。这就意味着我们可以完成这样一个矩阵（至少用一种方法）以获得合法的旅行（城市整个次序）。正如[131]中所描述的：

> "一个序列的布尔矩阵表达密封了所有有关序列的信息，包括单个城市到城市之间连接的微观拓扑关系及前任站和后继站的宏观拓扑关系。布尔矩阵表达可以用来理解现有的算子及开发的新算子能被应用于序列以产生想要的效果，同时保持此序列必要的特性。"

在[131]中，两个开发的新算子是交(intersection)和并(union)。两者都是类似杂交算子的二元算子。其他演化程序（如第 9 章中运输问题的 GENETIC-2），这样的算子应该在组合两个亲体的特征的同时遵守约束要求。

交算子是基于这样的观察：两个矩阵中位进行交导致矩阵：（1）1 的数目不大于 $\frac{n(n-1)}{2}$，（2）满足其他两个要求。这样，我们可以完成这样的矩阵来获得一个合法旅行（城市的整个次序）。

例如，两个亲体 $p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ 和 $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$由两个矩阵表达（图 10.2）：

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	1	1	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	1	0	1	0	0	0	0

图 10.2　两 个 亲 体

这两个矩阵的交得到的矩阵如图 10.3 所示。

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

图 10.3 交算子的第一阶段

由交结果强加的部分次序要求是：城市 1 优先于城市 2, 3, 5, 6, 7, 8, 9；城市 2 优先于城市 3, 5, 6, 7, 8, 9；城市 3 优先于城市 5；城市 4 优先于城市 5, 6, 7, 8, 9；城市 6, 7, 8 优先于城市 9。

交算子的下一步骤，其中一个亲体被选择；一些 1（对该亲体是特有的）被“加入”，矩阵通过分析行和列之和完成一个序列。如图 10.4 中的矩阵是完成第二步骤后可能的结果；它表示旅行 (1 2 4 8 7 6 3 5 9)。

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	0	0	0	1
4	0	0	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	0	0	0	0	0	0	0

图 10.4 交的最终结果

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	x	x	x	x	x
2	0	0	1	1	x	x	x	x	x
3	0	0	0	1	x	x	x	x	x
4	0	0	0	0	x	x	x	x	x
5	x	x	x	x	0	0	0	0	0
6	x	x	x	x	1	0	0	0	1
7	x	x	x	x	1	1	0	0	1
8	x	x	x	x	1	1	1	0	1
9	x	x	x	x	1	0	0	0	0

图 10.5 并算子的第一阶段

并算子是基于这样的观察：一个矩阵的位子集可以安全地与另一个矩阵的位子集合

并，只要这两个子集的交为空。该算子将城市集合分割成两个分离的组（在[131]中使用了一个特殊的方法以获得这种分割）。第一组城市是从第一个矩阵中拷贝位；第二组城市是从第二个矩阵中拷贝位。最后，它通过分析行和列的和将矩阵变成一个序列（如交算子）。

例如，两个亲体 p_1 和 p_2 分割城市成{1, 2, 3, 4} 和 {5, 6, 7, 8, 9}，产生的矩阵如图10.5 所示。就像是交算子完成的那样。

对城市的不同拓扑的实算结果（随机、群集、同心圆 — random, clusters, concentric circles）揭示了并和交算子的一个有趣的特性，不使用精华选项（ elitism option — 保存最好解），它也能使过程进步。这对 ER 或者 PMX 算子都是不可能的情况。几个对二元和一元（换 —swap、片 —slice 和转 — invert）算子的性能、复杂性和执行时间的整体的比较在[131]中有论述。

使用矩阵表达的第二个方法是由我的硕士生 David Seniw[353]描述的。当且仅当旅行直接从城市 i 到城市 j 时，行 i 和列 j 的矩阵元素 m_{ij} 包含一个 1。这就意味着矩阵中的每行和每列只有一个非零通道（对每个城市 i，精确地只有一个城市先于 i 被访问，也精确地只有一个城市紧接着 i 被访问）。例如图 10.6 (a)中所描述的一个染色体表示以这样的次序(1, 2, 4, 3, 8, 6, 5, 7, 9) 访问城市的旅行。值得注意的是：这种表达避免了指定起始城市所产生的问题，即图 10.6 (a)同时表示了旅行 (2, 4, 3, 8, 6, 5, 7, 9, 1) 和 (4, 3, 8, 6, 5, 7, 9, 1, 2) 等等。

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0	0	0

(b)

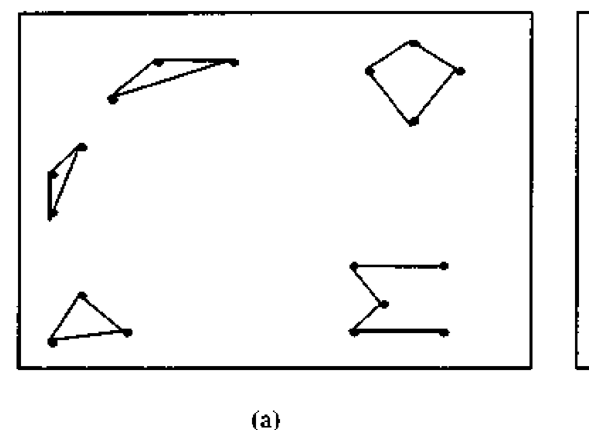
图 10.6 二进制矩阵染色体

值得注意的是每一个完整的旅行表达成二进制矩阵，其中每行只有一位和每列只有一位设定为 1。但是，不是每一个有这样性质的矩阵都表达一个单个的旅行。二进制矩阵染色体可以表达多个子旅行。每个子旅行将最终和自身构成一个循环，而不与染色体中任何其他子旅行相连。例如，图 10.6 (b)中的染色体表示两个子旅行 (1, 2, 4, 5, 7) 和 (3, 8, 6, 9)。

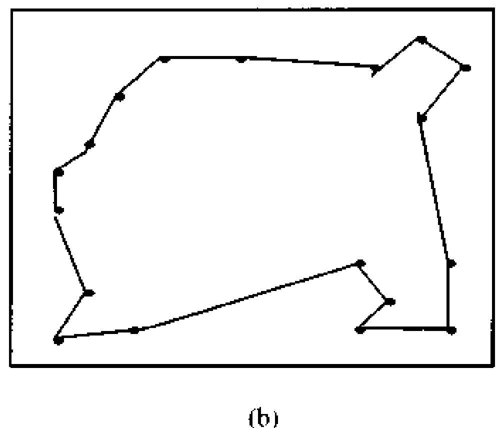
子旅行被允许是希望自然的群集发生。演化程序终止后，最好染色体通过使用一确定性算法陆续地组合成对子旅行而减缩为一单个的旅行。一个城市的子旅行（只剩一个城市，旅行返回其本身的旅行），其距离费用为零，是不允许的。一个低限 $q = 3$ 个城市

在子旅行中被设定，以图阻止 GA 将一个 TSP 问题减成大量的子旅行，而每个子旅行只有很少的城市（q 为此方法的一个参数）。

图 10.7(a)描绘了由该算法一试样运行于一定数量的城市其中有意放进群集所产生的子旅行。正如所预测的，该算法显现了分离的子旅行。图 10.7(b)描绘了子旅行组合后的旅行。



(a)



(b)

图 10.7 分离的子旅行和最后的旅行

定义了两个遗传算子：变异和杂交。变异算子取一个染色体，随机地在此染色体里选择几行和几列，移走这些行和列的相交点，并随机地在不同的形相里以可能的方式替换它们。

例如，考虑图 10.6 (a)的旅行，表达旅行：

(1, 2, 4, 3, 8, 6, 5, 7, 9)

假定行 4, 6, 7, 9 和列 1, 3, 5, 8, 9 随机地被选择以参与变异。这些行和列的边缘之和被计算。在这些列和行相交的位被移走并随机地替换，但它们必须与原来的边缘之和一致。换句话说，对应于取自原始矩阵的行 4, 6, 7, 9,和列 1, 3, 5, 8, 9 的子矩阵（图 10.8 (a)）被另一个子矩阵（图 10.8 (b)）替换。

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
1	0	0	0	0

(a)

0	0	1	0	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0

(b)

图 10.8 变异前(a)和变异后(b)的部分染色体

导出的染色体是用两个子旅行表示的一个染色体：

(1, 2, 4, 5, 7) 和 (3, 8, 6, 9)

如图 10.8 (b)所示。

杂交算子开始于一子体染色体，其中所有的位都重置为零。此算子首先检测两个亲体染色体，当发现两个亲体里有相同的位（同样的行和列）设置（即 1），则在子体中设置一个对应的位（阶段 1）。然后此算子交替地从每个亲体中拷贝一个设置位，直到在两个亲体中都没有不违反染色体构造基本限制可以拷贝的位存在（阶段 2）。最后，如果在子体染色体中的任何行仍然不包含一个设置位，该染色体将被随机地填入（最后阶

段）。 正如产生两个子染色体的经典杂交， 此算子随着亲体染色体的调换被执行第二次。

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	1	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

(b)

图 10.9　第一个(a)和第二个(b)亲体

下面的杂交的例子开始于图 10.9(a)里的第一个亲体染色体，它表达两个子旅行：

(1, 5, 3, 7, 8) 和 (2, 4, 9, 6)

第二个亲体染色体（图 10.9 (b)）表示一单个旅行：

(1, 5, 6, 2, 7, 8, 3, 4, 9)

建立第一个后代的前两个阶段如图 10.10 所示。

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0.	0	0
9	0	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

(b)

图 10.10　经过第一阶段(a)和第二阶段(b)杂交后的后代

经过杂交、在最后阶段以后的第一个后代如图 10.11 所示。它表示一个子旅行：

(1, 5, 6, 2, 3, 4, 9, 7, 8)

第二个后代表示

(1, 5, 3, 4, 9) 和 (2, 7, 8, 6)

注意在两个后代中有亲体染色体的共同片断。

此演化程序在从 30 个城市到 512 个城市的几个测试例子中都得到了合理的执行结

果。但是，不清楚的是参数 q（在一个子旅行中最小的城市数）对最终结果质量的影响。还有，组合几个子旅行到一单个旅行的算法远不显著。另外，这种方法和 Litke 的递归群集算法(recursive clustering algorithm)[246]有一些相似之处，此递归算法递归式地用单个有代表性的城市替换大小为 B 的群集，直到剩下的城市数小于 B 为止。因此，较小的问题可以被优化地解决。所有的群集被一个一个膨胀，算法排列当前旅行中的两个邻居之间的膨胀集。另外此算法对解决多个货郎担问题是有用的，其中几个货郎各自完成其单独的（非重叠的）旅行。

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0

图 10.11 经过最后阶段后的杂交后代

第三个基于矩阵表达的方法最近由 Homaifar 和 Guan[194]建议。如同先前的方法，当且仅当城市 i 到城市 j 有一条边，二进制矩阵 M 中的 m_{ij} 个元素被设定为 1。但是，他们使用了不同的杂交算子和启发式倒置 — 我们将依次讨论它们。

矩阵杂交(MX)算子被定义[194]。这些算子交换两个亲体矩阵中的所有表值，要么在单个杂交点（1–点杂交）之后要么在两个杂交点（2–点杂交）之间。另外的“修补算法”被运行：（1）移去重复，即确保每行和每列中只有一个 1，（2）切割并连接循环（如果有）以产生一个合法的旅行。

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0

(b)

图 10.12 带记号杂交点的二进制矩阵染色体

2 点杂交用下面例子说明。 两个亲体矩阵如图 10.12 所示：它们表示两个合法的旅

行：

(1 2 4 3 8 6 5 7 9) 和 (1 4 3 6 5 7 2 8 9)

两个杂交点被选择；它们是列 2 和 3 之间的点（第一个点），及列 6 和 7 之间的点（第二个点）。杂交点纵向切割矩阵；对每个矩阵，前两个列构造了分区的第一个部分，列 3、4、5、6 构成了中间部分，最后三列构成了第三部分。

经过 2 点 MX 算子的第一步后，两个矩阵中杂交点之间的表值都被交换（即，列 3，4，5，6 中的表值）。中间结果在如图 10.13 给出。

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	1
9	1	0	0	0	0	0	0	0	0

(b)

图 10.13　MX 算子第一步后的两个中间后代

两个后代(a)和(b)都是非法的；不过，每个中间矩阵中的 1 的总数是正确的（即 9）。“修补算法”的第一步按下面方式从矩阵中移动一些 1：每行和每列精确地有一个 1。例如，在图 10.13(a)中的后代中，重复的 1 发生在行 1 和行 3。此算法可移动表值 $m_{14} = 1$ 到表值 m_{84} 里，表值 $m_{38} = 1$ 到 m_{28} 里。类似地，在另一个后代里（图 10.13 (b)），重复的 1 发生在行 2 和行 8。此算法可移动表值 $m_{24} = 1$ 到表值 m_{34} 里，表值 $m_{86} = 1$ 到 m_{16} 里。经过此修补算法的第一步后，第一个后代表示一个合法的旅行：

(1 2 8 4 3 6 5 7 9)

第二个后代表示由两个子旅行组成的旅行：

(1 6 5 7 2 8 9) 和 (3 4)

修补算法的第二步应该只用于第二个后代。在这一阶段。此算法切割并连接子旅行以产生一个合法的旅行。切割和连接步骤涉及的是原始亲体中已存在的边。例如边 (2 4) 被选择连接这两个子旅行，因为这条边存在于一个亲体中。这样，完整的旅行（合法的第二个后代）为：

(1 6 5 7 2 4 3 8 9)

Homaifar 和 Guan[194]使用的第二个用来补充 MX 杂交的算子是探索启发式倒置。该算子颠倒两个切割点之间的城市次序（正如简单的倒置那样 — 我们在本章的前面曾讨论过）。如果两个切割点之间的距离很大（高阶倒置），此算子开拓了“好”路径之间的连接，否则（低阶倒置）此算子执行局部搜索。不过，经典的和建议的倒置算子之间有

两个不同。第一个是只有在新的旅行好于原始的旅行时，所导出的后代才被接受。第二个不同是倒置过程在旅行中选择一个城市并检查（最低可能的）阶 2 倒置的改进。若第一个倒置得到的改进被接受，倒置过程终止，否则，考虑阶 3 的倒置，等等。

报告的结果[104]表明带 2 点 MX 和倒置算子的演化程序成功地用在 30~100 个城市的 TSP 问题。在最近的实算中，此算法对有 318 个城市的 TSP 问题的结果只偏离最优解 0.6%。

在本章，我们不给出对不同数据编码及“遗传”算子的各种实算的精确的结果。而是对许多尝试建立针对 TSP 成功的演化程序做一个总的综述。理由之一是多数引用的执行结果都严重地依赖于许多细节（群体规模、代数、问题大小等等）。而且，许多结果只是针对相对小的 TSP（至多 100 个城市），正如[207]中所评述的：

“看起来，100 个城市的例子现在可以被认为在全局最优化技术世界里是恰到好处，对我们远大于此数的例子毫无疑问地必须使用启发式方法。”

不过，本章引用的多数文献对各自提出的方法和其他方法进行了比较。对这些比较，使用了两类测试实例：

- 随机挑选的城市。这里，一个期望最小 TSP 旅行的长度 L^* 的经验公式是有用的：

$$L^* = k\sqrt{n \cdot R}$$

  其中 n 为城市数，R 为方盒子的面积，此盒中的城市是随机放置的，k 为一个经验常数，约等于 0.765①（见[372]）。
- 公共可用的城市集（部分伴随最优解），见网址 ftp softlib.rice.edu 的目录 /pub/tsplib。其中有两个文件 tsplib.sh 和 tsplib.tar; 分别 6MB 和 2MB。

为得到遗传算法技术用于 TSP 的完整的轮廓，我们应该报告其他研究者的工作，如[288]和[389]，他们将 GA 用于 TSP 的局部优化上。局部优化算法（2-opt，3-opt，Lin–Kernighan）十分有效。正如[207]中所说的：

“倘若问题是 *NP* 难解问题，因此发现最优旅行的多项式乘算法(polynomial-time algorithms)是不大可能存在的，更多注意力应放在只要求找到次优旅行的有效近似算法、快速的算法。时至今日，实用中最好的这样的算法是基于（或者起源于）被称为局部优化的一般技术，其中一个给定解通过局部变化被迭代改进。”

局部优化算法，对一给定（当前的）旅行，指定相邻旅行集合并用一（可能）更好的邻居替换当前旅行。这种步骤被应用，直到达到局部最优解。例如，2-opt 算法，只通过修正两条边，而从另一个旅行获得来定义相邻的旅行。

局部搜索算法是开发遗传局部搜索算法的基础[389]，其步骤为：

- 通过一个局部最优的旅行用局部搜索算法来替换当前群体（大小为 μ）中的每个旅

① 根据 David Johnson[209]，这样的实算比较应该非常仔细地做。首先，渐近常数的估算略低，更像是 0.7128 (±0.0005)。第二，比率 $L^*\sqrt{n}$（对 $R=1$）收敛得非常慢——对 $n=100\,000$，它为 0.7134，对 $n=1000$ 它有点像 0.7306（两种估算都来自于 David Johnson[209]）。第三，对较小规模，例子之间的变化相当大。对 $n=1000$，标准偏差为 0.0064 左右，对 $n=100$ 为 0.0224。结论是对给定的例子，应该比较旅行长度和 Held–Karp 边界，而不是期望优化值的估算。Held–Karp 较低的边界[179,180]要求涉及评价几个生成树(spanning trees) 紧随着 Lagrange 松弛因子的迭代过程。

行。

- 通过另外λ个旅行扩展群体——应用于当前群体中的一些旅行重组算子的后代。
- （再次）使用局部搜索算法，用一个（局部最优的）旅行替换扩展群体的 λ 个后代中的每一个。
- 根据一些选择规则（适者生存），减少扩展群体至其原始的规模 λ 。
- 重复最后三步，直到满足停止规则（演化过程）。

注意，遗传局部搜索算法和$(\mu+\lambda)$演化策略（第 8 章）之间的一些相似之处。和$(\mu+\lambda)$-ES 一样，μ个个体产生λ个后代，且新的（扩展的）有$(\mu+\lambda)$ 个个体的群体通过选择过程再次减小至μ个个体。

上述遗传局部搜索算法[389]和早先由 Mühlenbein，Gorges–Schleuter 和 Kramer[288]建议的对 TSP 的遗传算法相似，它鼓励个体的“智能进化”。此算法

- 使用局部搜索算法（二优）以用一（局部最优）旅行替换当前群体中的每个旅行，
- 选择亲体配对（平均之上的个体得到更多的后代），
- 再生（杂交和变异），
- 用每个个体搜索最小值（减少、问题求解器、扩展），
- 重复最后三步，直到满足停止条件（演化过程）。

在该算法中使用的杂交是一个变种的次序杂交（OX）。这里，两个亲体（两个切割点用“|”标记），

$$p_1=(1\,2\,3\,|\,4\,5\,6\,7\,|\,8\,9)$$
$$p_2=(4\,5\,2\,|\,1\,8\,7\,6\,|\,9\,3)$$

按照下面的方式产生后代，首先，切割点之间的片断被拷贝到后代中：

$$o_1=(x\,x\,x\,|\,4\,5\,6\,7\,|\,x\,x)$$
$$o_2=(x\,x\,x\,|\,1\,8\,7\,6\,|\,x\,x)$$

往下（和 OX 的例子不同，不是从一个亲体的第二个切割点开始），来自另一个亲体的城市被以相同的次序从串的开始被拷贝，并略去已存在的符号：

$$o_1=(2\,1\,8\,|\,4\,5\,6\,7\,|\,9\,3)$$
$$o_2=(2\,3\,4\,|\,1\,8\,7\,6\,|\,5\,9)$$

实算结果还是令人鼓舞的。此算法找到了 532 个城市的问题的一个旅行，找到的长度为 27702，这和最优解（27686, 由 Padberg 和 Rinaldi[302]发现）只差 0.06%。

另有两个其他方法最近被报道。第一个方法是 Craighurst 和 Martin[68]建议的，他们集中于开发近亲预防（见第 4 章）和解 TSP 遗传算法性能之间的联系。作者在对它们的实算中使用了具有下面特征的一个 GA：群体规模为 128、基于分级加权的代的选择、其中 128 个后代与 128 个父代竞争，以权重为基础的 MPX 杂交，局部爬山法（二优）作用于后代生成，变异率为 0.005，终止条件是 500 次连续的代没有改进。近亲繁殖的方法是面向家族的：作者只关心被选择杂交的两个个体的先辈，并引入了几个繁殖法则。第 k 个近亲繁殖法则禁止个体与 k–1 个先辈配对（即，对 $k=0$，没有限制；对 $k=1$，一个个体不能与其本身配对；$k=2$，它不能与其本身、其父母、其孩子、更不能与其兄弟姐妹配

对等等）。并做了几个实算（城市数的范围在 48~101 的测试问题，取自本章前面提到的测试库 softlib.rice.edu）。结果表明近亲禁止法则和变异率有着强烈的相互依赖性：对低变异率，近亲繁殖阻止了结果的改进，但是，如果变异率升高，近亲繁殖阻止机制的重要性就会降低，直到损害系统的结果（对高变异率 ≈ 0.1）。还有，不同的近亲繁殖阻止法则并不以显著的方式影响群体的多样性（这里两个个体之间的相似性被一个比率度量，该比率为解的总边数和解之间共享的共同边数之差与解的总边数之比）。最终的结论是对下面问题 “是否禁止要更好？” 的一个否定的回答。有关结果更详细的讨论，见[68]。

Valenzuela 和 Jones[390]建议了一个有趣方法以应用演化算法到艰难的组合问题；他们的方法是基于对 TSP 的 Karp–Steele[219,379]算法技术中的分隔并征服(divide and conquer)。他们的演化分隔并征服（EDAC）算法可用在任何这样的问题：其中有关子问题的较好解的知识在构造一个全局解时是有用的，而且他们应用这种技术到几个 TSP 问题。其中几个对分法可以考虑；这些方法将有 n 个城市的矩形切割成两个较小的矩形（譬如分割问题的方法之一是用精确的平行于较短边的线对分矩形的面积；另一个方法是横穿 int(n/2)个到矩形较短边最近的城市，这样就在两个子矩形之间提供了一个“共享”城市）。最终的子问题十分小（基本是在 5 和 8 个城市之间），它是相当容易解的（2-opt 被选择处理这种情况，因为它快而简单）。补片算法(patching algorithm)替换两个分开的旅行中的某些边以得到一个更大的旅行。现今选择遗传算法的主要作用是确定在每个阶段的对分方向（横向或者纵向）。值得注意的是：个体染色体表达使用的数据编码是一个 $p \times p$ 二进制① 数组 M，它和 TSP 方形的几何区域相关。如果在分隔并征服算法的某个阶段，一个矩形被对分，从矩阵 M 最靠近相应矩形中心的一位就被选择；该位的值确定当前切割的方向（横向对纵向）。在[390]中使用的遗传算子是直接的：杂交交换两个数组之间的二进制元素，数组沿着 x 轴或者 y 轴被切割② （在两点）。变异以常概率触发数组的所有位（使用 0.1。即数组中 10%的位变异）。有关实算结果及遗传算法在这种方法中全面的贡献，见[390]。

看起来一个针对 TSP 的好的演化程序应该合并局部改进算子（变异类型），算子是基于局部优化算法的，还要有精心设计的并入有关问题的启发性信息的二进制算子（杂交组），本章我们用一个简单的评述总结：对包含“最好”表达和操作它们的“遗传”算子的 TSP 演化程序的探索，还在继续之中！

---

① 作者考虑一个方形区域，因此矩阵有相同的维数。

② 作者使用了一个附加的限制，即沿着轴这两个切割点之间的距离在三分之一和三分之二之间以确保对某个亲体的遗传材质的合理分割。

第 11 章　基于各种离散问题的演化程序

正如在引言部分所陈述的那样，看起来多数研究者通过使用非标准的染色体表达，和/或通过设计与问题有关的遗传算子（如[141]、[385]、[65]、[76]等等）以使之适用于要解决的问题来修正它们的遗传算法的实现，这样就建立了有效的演化程序。这样的修正在前面两章里分别对运输问题和货郎担问题进行了详细的讨论（第 9 章和第 10 章）。在本章里，我们有些随意地选择了几个由作者和其他研究者开发的演化程序，它们都是基于非标准的染色体表达和/或与问题知识有关的算子。我们将讨论日程表问题（scheduling problems ，11.1 节）、时间表问题（timetable problem，11.2 节）、分割问题（partitioning problems，11.3 节）及在移动式机器人环境中的路径安排问题（path planning problem，11.4 节）。本章用单独的 11.5 一节进行总结，对其他几个有趣问题进行了简评。

描述的系统及它们应用的结果提供了支持演化规划方法的更多的论证，它促进了数据编码结合一类特殊问题算子的产生。

11.1　日 程 表

工作车间展示了一个工艺过程——有组织地制造设备；其主要特征是要完成工作的多样性[182]。一个工作车间生产物品（部件）；这些部件有一个或者多个可供选择的工艺过程计划。每个工艺过程计划由一个序列操作组成；这些操作要求资源和一定的预定机器工作时间。一个工作车间处理命令，其中的每个命令是针对一定数目的同种部件的。计划、日程表及控制工作的任务是非常复杂的，在这些任务中只有有限的分析过程[182,132]可获得帮助。

工作车间的日程表问题是选择操作序列以协调开始/结束时间和每种操作资源。主要要考虑的是处理闲置机器和劳动力的费用、处理存货的费用及在限期内完成一定的命令的需要。正如[182]中所描述的：

> "不幸的很，这些考虑彼此趋向于冲突。可以只提供最小数量的机器和人力来使闲置的机器和劳动力的费用最少。但这将导致较多的工作处于等待状态，因此造成大量的待处理存货及难以在限期内完成任务。另一方面，可以通过提供较多的机器和劳动力来使命令不会常常等待被处理以根本保证在期限内完成任务。但这将会导致对闲置机器和劳动力过多的费用。因此，在这些考虑之间做一个折衷是必要的。"

有各种各样的工作车间日程表问题，每个都有一些特定的约束（如维修、机器损耗及准备时间等等）为特征。

让我们考虑一个简单的工作车间问题例子来阐明上述的描述。

例 11.1 假定有三个命令，o_1，o_2，和 o_3。对每个命令，部件(parts)数量和生产单元号为：

o_1 : $30 \times$ part a

o_2 : $45 \times$ part b

o_3 : $50 \times$ part a

每个部件有一个或者多个可供选择的处理方案(plan)：

a : plan # 1_a (opr_2, opr_7, opr_9)

a : plan # 2_a (opr_1, opr_3, opr_7, opr_8)

a : plan # 3_a (opr_5, opr_6)

b : plan # 1_b (opr_2, opr_6, opr_7)

b : plan # 2_b (opr_1, opr_9)

其中 opr_i 项表示要求被执行的操作(operation)。每个操作要求一定的在一台或多台机器的工作时间；它们是：

opr_1 : (m_1 10) (m_3 20)

opr_2 : (m_2 20)

opr_3 : (m_2 20) (m_3 30)

opr_4 : (m_1 10) (m_2 30)(m_3 20)

opr_5 : (m_1 10) (m_3 30)

opr_6 : (m_1 40)

opr_7 : (m_3 20)

opr_8 : (m_1 50) (m_2 30) (m_3 10)

opr_9 : (m_2 20) (m_3 40)

最后，每个机器有其必要的改变操作的准备时间：

m_1 : 3

m_2 : 5

m_3 : 7

遗传算法界对工作车间问题颇感兴趣。解决此问题的最早尝试之一是由 Davis[72]报告的。其方法的主要思想是按这样的方式对日程表中的表达进行编码：（1）遗传算子将以一种有意义的方式被操作；（2）解码器总是产生对问题的合法解。这种对操作的解编码并在评价时解码的策略是十分普遍的，并可以用在各种各样的约束问题 — 同样的思想被 Jones[211]用来解决分割问题（见 11.5 节）。

总之，我们是要表达日程表的信息，如"机器 m_2 从时间 t_1 到时间 t_2 里，在部件 a 上执行操作 o_1"。但是，多数使用这样信息的算子（变异、杂交）将产生非法的日程表 — 这就是为什么 Davis[72]使用一个编码/解码策略的原因。

让我们看一看编码策略是如何被应用于工作车间问题的。该系统是由 Davis[72]通过保持一个对每个机器的优先表来开发的；这些优先权是和时间相连的。该表的一个初始成

员是该表起作用的时间，表的其余部分是一些命令排列、加上两个另外的元素："等待"和"闲置"("wait" and "idle")。解码过程以这样的方式模拟任务的操作：无论什么时候，一旦一个机器准备好作挑选，其优先表中的第一个被允许的操作将被采用。所以如果对机器 m_1 的优先表是：

$$m_1 \ : \ (40\ o_3\ o_1\ o_2\ \text{'wait'}\ \text{'idle'})$$

那么，在时间 40 的解码过程将从机器 m_1 的命令 o_3 中搜索一个部件来加工。如果不成功，解码过程将从命令 o_1 和 o_2（即先从 o_1；如果失败，则从 o_2）搜索一个部件。 这样的表达保证了一个合法的日程表。

算子是与问题有关的（它们从确定性方法中衍生出来）：

- 运行–闲置 (run–idle) 此算子只应用于已等待一个小时以上的机器的优先表。它插入"idle"作为优先表的第二个成员，并重置优先表的第一个成员时间到 60（分钟）；
- 搅混(scramble) 此算子"搅混"优先表中的成员；
- 杂交(*crossover*) 此算子为选择机器交换优先表。

对搅混和杂交，算子的概率从运行开始时从 5%和 40%下降到后来的 1% 和 5%。运行–闲置概率被设定为机器用在等待的时间除以总模拟时间的百分数。

不过，实算是在两个命令、六台机器、三个操作[72]的小例子上进行的，所以很难评价此方法的可用性。

另一组研究者是用 TSP 的观点研究工作车间问题的[62, 383, 384, 402]。 动机是多数为 TSP 开发的算子都是"盲目的"，即它们不使用任何有关城市之间的实际距离的信息（见第 10 章）。这就意味着这些算子在其他序列问题上可能是有用的，因为这些问题中的两点之间没有距离（城市、命令、工作，等等。）。但不一定是如此情况。虽然 TSP 和日程表问题这两个问题都是序列问题，但它们展示了不同的与问题有关的特征。对 TSP，重要的信息是城市的相邻信息，而在日程表问题里，项的相对次序是主要关心的。相邻信息对日程表问题是无用的，而相对次序对 TSP 是不重要的，缘于旅行的循环性质：旅行 (1 2 3 4 5 6 7 8) 和 (4 5 6 7 8 1 2 3)实际上是等同的。这就是为什么对不同的应用，我们需要不同的算子。正如[370]中的评述：

> "Gil Syswerda[383]指导了一项研究，其中"边重组"（一个特别为 TSP 设计的遗传算子）在工作序列日程表任务中相对其他算子执行得要差。而 Syswerda 使用的群体规模较小（30 个串），且只使用变异（没有重组）反而对此问题获得较好的结果，Syswerda 对不同序列任务中有关位置、命令和邻位的相对重要性的讨论引发了一个还没有充分证明的论点。包括我们[402,403]在内的研究者似乎默认所有的序列任务都是相似的，一个遗传算子对任何类型的序列问题应该是足够了。"

相似的评述一年前由 Fox 和 McMahon[131]得出① ：

① 1991，重印得到 Rawlins, G., *Foundations of Genetic Algorithms* 的许可。

"主要关心的是每个遗传算子对各种序列问题的应用性。例如，在 TSP 里，序列的值等价于颠倒次序后该序列的值。这种特征并不是对所有的序列问题都是正确的。对日程表问题，这将是一个严重的错误。"

在[370]中，六个序列算子（次序杂交：order crossover、部分映射杂交：partially mapped crossover、循环杂交：cycle crossover、增强边重组：enhanced edge recombination、基于次序的杂交：order–based crossover，及基于位置的杂交：position–based crossover，所有这些算子都在前面的章节中进行过讨论）在两个不同的序列任务上进行了比较：一个 30 个城市（盲目的）TSP 及一个 195 个元素的日程表应用序列任务。正如所预料的，日程表优化的结果（就六个算子中"最适"而言）几乎和 TSP 的结果相反。在日程表优化例子中，增强边重组算子最好，紧随其后的是次序杂交、基于次序的杂交及基于位置的杂交，PMX 和循环杂交最差。而在 TSP 中，最好的是基于位置的杂交和基于次序的杂交，随后是循环杂交和 PMX，次序杂交和增强边重组最差。这些不同可以通过检测这些算子如何保存邻位（对 TSP）和次序（对日程表问题）信息得到解释。

对其他序列（次序）问题可得到相似的评述。在[78]中，Davis 描述了一个基于次序的遗传算法来解决下面的图着色问题：

"给定一个带有加权结点和 n 个颜色的图，分配颜色给结点以达到最高的分数，规定没有成对的（即直接用线相连的结点）可以有同样的颜色；得分为着色结点的总的权值。"

一个简单的贪婪算法(greedy algorithm)按递减的权值排列此结点集并按照这样的次序处理结点（即从颜色表中分配第一合法的颜色给结点）。很明显，这是一个序列问题，至少一个结点的排列将返回最大的利益值，所以我们将搜索优化的结点序列。另一点也很明显，简单的贪婪算法不能保证最优解：还须使用一些其他的技术。表面上，此问题类似 TSP，在那里我们刚刚跟从货郎访问城市的最好次序。但是此问题的"本质"是非常不同的：如，在图着色问题里，结点有权值，而 TSP 中权值是在结点之间分配（即距离）。在[78]中，Davis 以一个结点表表达一个次序（如 (2 4 7 1 4 8 3 5 9)，作为 TSP 中的路径表达），并使用了两个算子：基于次序的杂交（在前面章中 TSP 中使用的算子讨论过）和搅混子表变异(scramble sublist mutation)。即使是变异，它也应当对染色体进行局部修正，似乎与问题有关[78]：

"有一种诱惑想把变异作为染色体两个区域值的交换。我已经将它尝试到几个不同的问题上，但是依我看，它并不比称为'搅混'子表变异的算子工作得好。"

搅混子表变异从一个亲体中选择结点的一个子表，并将它杂混在后代里。亲体（被选择子表的开始和结尾用"|"标记）：

$$p = (2\ 4\ |\ 7\ 1\ 4\ 8\ |\ 3\ 5\ 9)$$

可产生后代

$$o = (2\ 4\ |\ 4\ 8\ 1\ 7\ |\ 3\ 5\ 9)$$

但该算子对其他次序或者日程表问题的执行情况尚须保留。我们再一次引证[78]中的话：

“许多其他类型的变异可以用在基于次序的问题。搅混子表变异是我常使用的最普通的一个。时至今日，还没有有关这类算子的文章出版，尽管这是未来工作中一个有希望的专题。”

现在回到日程表问题。如前所述，Syswerda[383]开发了一个针对日程表问题的演化程序。但挑选的是一个简单的染色体表达：

“在挑选对……日程表的染色体表达时，我们有两个基本的元素可以选择。第一个是被纳入日程的任务表。该表非常像 TSP 问题被访问的城市表。……一个使用任务序列的选择是直接用日程表作染色体。这看起来像是一个过分笨拙的表达，需要复杂的算子对其进行操作，但当处理复杂的现实世界如日程表问题时，却有决定性的优势。……我们的情况是求助于一个简洁的染色体表达，而不是一个复杂的表达。对日程表问题，我们使用的染色体结构上和上述 TSP 中的一样，但是我们将使用任务次序，而不是城市。”

染色体可以用一个日程表构造器(schedule builder)来解释，日程表构造器是一段用以“理解”日程表任务细节的软件。这种表达以特殊的算子支持。三个变异被考虑：基于位置的变异（两个任务是随机的，第二个任务被放在第一个任务之前）、基于次序的变异（两个被选择的任务随机地被交换）和搅混变异（和 Davis 的搅混子表变异相同，在前面段落中有叙述）。所有这三个变异都比随机搜索执行得好，基于次序的变异明显略胜一筹。如前所述，对日程表问题最好的杂交算子是基于次序的和基于位置的杂交。

看起来，简单表达的挑选不是最好的。从其他不相关的实算如运输问题（第 9 章）判断，我们感到染色体表达更有利于日程表问题。确实，在这种情况下，必须下大力气来设计与问题有关的“遗传”算子：然而这种努力的回报是增加速度和改进系统的性能。而且，一些算子可能不是十分简单[383]：

“一个运行于日程表的简单贪婪算法通过移走一个低质量的任务或者两个并用高质量的任务替换它们就可能找到一个高质量的任务。”

我们相信，总的来说，对实际的日程表问题，跟从的方向是：不仅仅在算子中并入与问题有关的知识（如简单的染色体表达那样），同时也并入染色体编码结构中。应用这种方法的第一个尝试已经出现。在 Husbands、Mill 和 Warrington[198]的研究中，他们将染色体表达成一个序列：

$$(opr_1\ m_1\ s_1)\ (opr_2\ m_2\ s_2)\ (opr_3\ m_3\ s_3)\ \ldots$$

其中，opr_i, m_i, and s_i 分别表示第 i 个算子、机器及准备。

在[21]中，作者比较了三种表达，从最简单的（表达-1）：

$$(o_1)\ (o_2)\ (o_3)\ \ldots$$

到中间的（表达-2）：

$$(o_1 \text{ plan } \# \ 1_a)\ (o_2 \text{ plan } \# \ 2_b)\ (o_3 \text{ plan } \# \ 2_a)\ \ldots$$

再到最复杂的（表达-3）：

$$(o_1 \langle opr_2 : m_2,\quad opr_7 : m_3,\quad opr_9 : m_2 \rangle)\ (o_2 \langle opr_1 : m_3,\quad opr_9 : m_2 \rangle)$$

$(o_3\langle opr_1 : m_1,\quad opr_3 : m_2, opr_7 : m_3,\quad opr_8 : m_1\rangle)\ldots$

表达–3 的结果要远远好于其他两个表达。在总结性的讨论中，作者评述道[21]：

“算子本身必须被调节使之适合域的要求。染色体表达应该包含所有属于优化问题的信息。”

概括起来，有可能在染色体表达的基础上将所有基于遗传算法的方法归类为各种日程表问题。它们属于两类[21]：

- 间接表达： 其中从染色体表达到合法生产日程的转换必须用一个特殊的解码器（日程表构造器）完成；只有这样一个个体的解才能被评价。更进一步，这些表达可以被分成[53]域无关和与问题有关的表达，我们在早先的段落里已看到了这两种情况。
- 直接表达： 其中生产日程本身被当成染色体使用（如[198]）。通常，这种表达要求一些与问题有关的算子[53]。

有关对日程表问题的演化算法的完整的概述，可见[54]。

11.2 时间表问题

运筹学中最有趣的问题之一是时间表问题。时间表问题有重要的实际应用：它已被深入细致的研究并被公认是 *NP* 难解问题(*NP*–hard)[106]。

时间表问题合并了多种多样的非常规约束——这可能是为什么（就作者所知）直到最近才用遗传算法技术对此作第一次尝试[63]。时间表问题有许多版本；其中之一可以描述为：

- 教师表 $\{T_1,\ldots,T_m\}$
- 时间间隔表（小时） $\{H_1,\ldots,H_n\}$,
- 班级表 $\{C_1,\ldots,C_k\}$。

问题是找到最优的时间表（教师–时间–班级）：目标函数是要满足一些目标（软约束）。包括教学目标（例如在整个周传教一些班级）、个人目标（例如对某些兼职教师保证下午的自由），组织目标（例如每小时都有教师以备临时的教学任务）。约束包括：

- 每个教师和每个班级有预定的小时数，合法的时间表必须“符合”这些数，
- 每次每个班级只能有一个教师，
- 一个教师一次只能教一个班，
- 在某些时间间隙对每个班安排一个教师。

看起来，对时间表问题的一个潜在解最自然的染色体表达是矩阵表达：矩阵 R_{ij} ($1\le i\le m$, 及 $1\le j\le n$)，其中每行对应于一个教师，每列对应于一个小时；矩阵 R 的元素是班级（$r_{ij}\in\{C_1,\ldots,C_k\}$）①。

在[63]中，约束主要被遗传算子管理（作者还使用了一个修补算法来消除同时在同一

① 在[63]中，矩阵 R 的元素实际上是带三种可能下标的班级，这些下标包含部门、临时教学任务等概念。

时间同一个班出现多于一个教师的情况）。使用了下面的遗传算子：

- 阶 k 的变异：此算子从矩阵 R 的同一行里选择两个有 k 个元素的邻接序列，并交换它们。
- 日变异：(day mutation)此算子是前面算子的特殊情况：它选择矩阵 R 对应于不同日子的两列中的元素组（小时），并交换它们。
- 杂交：给定两个矩阵 R_1 和 R_2，算子按所谓的局部适应函数值的降序排列第一个矩阵中的行（适应值函数的一部分是专门针对每个教师的特性的结果），最好的 b 个行（b 是在局部适应值函数和其双亲基础上由该系统确定的一个参数）被抽取作为一个基因块；剩余的 $m-b$ 行从矩阵 R_2 中抽取。

导出的演化程序成功地测试了意大利米兰的一所大学校的数据[63]。

时间表问题也由 Paechter, Luchian 和 Petriuc[303]进行了研究，他们比较了两个演化方法（时空排列法 time-space permution method 和布局寻找法 place and seek method），对一个大的真实世界的时间表问题。最近，Burke 等[57]描述了一个用于高度约束的时间表问题的混合遗传算法。该方法组合了时间表直接表达和几个启发式杂交算子及启发式变异算子。这样，此算法用特殊的数据编码和算子维持了解的可行性。

11.3　分割对象或图

有一类分割① 问题，它要求将 n 个对象分割到 k 个类里。这包含许多知名的问题，像装箱问题（分配物品到箱子里），图着色问题（给一个图的结点分配具体的颜色）等等。对各种类型的分割问题，已开发出许多种不同的系统；在这一节，我们将对它们中的一些进行讨论。

一类演化程序是将所有的对象（例如装箱问题中的物品或者图着色问题中的结点）表达成一个排列表；特殊的算子可以被应用，一个解码器对分配做决策。例如对图着色问题，Davis[78]在一个染色体中表达了一个结点的排列并对此结构使用了特殊的算子（均匀的基于次序的杂交、基于次序的变异）。同时，一个贪婪解码器(greedy decoder)被用来解释此结构，它如下工作：考虑一个特定的颜色并用此颜色按染色体所给定的次序涂加在所有的结点上（如果可能）。当没有此颜色的着色可能时，转向下一个颜色。Davis[78]给出了对一个有 100 个结点的图着色问题的实算结果。

对分割问题的一个有趣的方法② 在[235]中提出。Von Laszewski 使用组数编码法(group-number encoding)编码分割，即，分割被表达成 n 个整数的串，

$$(i_1,\ldots,i_n)$$

其中，第 j 个整数 $i_j \in \{1,\ldots,k\}$ 表示分配给对象 j 的组数。支持此表达的是“智能结构算子”(intelligent structural operators)：结构杂交(structural crossover)和结构变异(structural mutation)。我们将依次讨论它们。

① 有时这些问题被称为分组问题 (grouping problems)[109]。

② 分割问题此版本的另一个要求是分割尺寸相等或者近似相等。

- 结构杂交：

结构杂交的机制可用下面的例子来解释。假定，有两个被选择亲体（12 个数的串）：

$p_1 = (112311232233)$

$p_2 = (112123122333)$

这些串被解码成下面的分割：

p_1： {1,2,5,6}, {3,7,9,10}, {4,8,11,12}

p_2： {1,2,4,7}, {3,5,8, 9}, {6,10,11,12}

首先，一个随机的分割被挑选：设为分割#2。该分割从 p_1 拷贝到 p_2 里：

$p_2' = (112123222233)$

拷贝过程（如上述例子所见）通常破坏了相等分割尺寸的要求，因此，我们应用一个修补算法。注意在原始的 p_2 中，有的元素分配给分割#2 而在被拷贝分割里没有这些元素：即元素 5 和 8。这些元素被抹去，

$p_2'' = (1121*32*2233)$

并由其他分割的数随机地替换，它被重写在拷贝步骤中。这样最后的后代可能是

$p_2''' = (112133212233)$

如前所述，在组数编码中，两个同等的分割可能被不同的串表达，缘于不同的分割数目。考虑到这些，在杂交被执行前，编码应适应两个亲体之间的差别最小化。

- 结构变异：

典型地，变异将用一些随机数替换一个串中单个成员；但是这将破坏切割相同大小的要求。结构变异定义串中两个数的交换。这样，一个亲体

$p = (112133212233)$

可产生下面的后代（位置 4 和 6 上的数被交换）：

$p' = (112331212233)$

算法是用附加的策略（如亲体替换策略）增强的一个平行遗传算法实现的；对一个有 900 个结点、最大结点度为 4 的随机图，这种演化程序比其他启发式算法有显著的杰出性能[235]。Mühlenbein[287]尝试了相似的方法，他实算了相同组数编码、同时也使用了“智能”杂交，它传送给整个分割而不是分离的对象。

Jones 和 Beltramo[211]构造了几个演化程序。这些程序使用了不同的表达和几个算子。有趣的是并入与问题有关的知识对已开发演化程序性能的影响。两个选择的测试问题是：

- 将 n 个数分成 k 个组以最小化组内和之间的差，
- 将美国本土的 48 个州分割成 4 个颜色组以使同一组中相邻州的成对数最小。

演化程序的第一组将分割编码成 n 个整数串

$(i_1,\ldots, i_n)$

其中，第 j 个整数 $i_j \in \{1,\ldots,k\}$ 表示分配给对象 j 的组数：这就是组–数编码。

组–数编码对使用标准的算子是可能的。变异将用$\{1,\ldots,k\}$中的一随机数替换一单个（随机选择的）基因 i_j。单点或均匀杂交将总能产生一合法后代。但是正如[211]中所指

出的，一个后代（经过变异或者杂交）可能包含小于 k 个组，而且，两个代表同一分割亲体的一个后代可能表达整个不同的分割，缘于不同的组号数。特殊的修补算法（拒绝法：rejection method、亲体的重编号：renumbering the parents）可以用来消除这些问题。同时，我们还可考虑使用基于边的杂交（在前面章中有定义）。这里，我们假定当且仅当它们在同一组里，两个对象被一条边相连。基于边的杂交通过从亲体中组合边构造了一个后代。

值得注意的是对两个测试问题的几个实算表明基于边的算子的优越性[211]；但使用的表达不支持此算子。正如所报告的，每次迭代，它比其他杂交方法花 2～5 倍的机时。这是由于不适当的表达造成的：例如，两个亲体

$$p_1 = (11222233)$$
$$p_2 = (12222333)$$

表示下面的边：

p_1 的边：(12), (34), (35), (36), (45), (46), (56), (78)

p_2 的边：(23), (24), (25), (34), (35), (45), (67), (68), (78)

一个后代应该至少包含亲体中的一条边，如，

(11222333)

代表下面的边：

(12), (34), (35), (45), (67), (68), (78)

但是，边的选择过程不是直接的：如(56) 和 (67)的选择——其中这两条边都在上面被表达——意味着边(57)的存在，而它现在并不在里边。

看起来一些其他的表达更适合该问题。演化程序的第二组将分割编码成 $n+k-1$ 个不同整数的串：

$$(i_1, \ldots, i_{n+k-1})$$

值域{1,..., n }中的整数表示对象，值域{n+1,...,$n+k$-1}中的整数表示分隔点：这就是分隔点排列(permutation with separators)编码。

例如，7–串

(1122233)

被表示为一个 9–串

(128345967)

其中 8 和 9 是分隔点。

当然，所有 k-1 分隔点都必须被用到；同样，它们不能出现在第一和 / 或最后的位置上，它们也不能连在一起——一个挨着一个出现（否则，一个串将解码成小于 k 个组）。

照例，在设计算子时应该小心。这些和解货郎担问题时使用的算子相似，其中 TSP 被表达成一个城市的排列。变异将交换两个对象（分隔点除外）。两种杂交被考虑使用：次序杂交(order crossover, OX)和部分匹配杂交(partially matched crossover, PMX) ——它们已在第 10 章中讨论过。杂交将被重复操作直到后代解码成有 k 组的一个分割①。

① Jones 和 Beltramo[211]每次杂交只产生一个后代。

通常，基于分隔点排列编码的演化程序结果比基于组–数编码程序好。但是，两种编码方法都没有较多的使用与问题有关的知识。可以构造并入目标函数知识的第三族演化程序。在[211]中，这种方法是按下面的方式做的。

使用的表达是最简单的一种：每个 n–串表示 n 个对象：

$$(i_1, \ldots, i_n)$$

其中 $i_j \in \{1, \ldots, n\}$表示对象数。因此对 $j \neq p$，$i_j \neq i_p$ 。对此表达的解释使用了一个贪婪启发式规则：串中前 k 个对象被用来初始化 k 个组，即前 k 个对象的每一个都被放在一分离的组里。其余的对象以先入先出的基准被加入，即它们以串中出现的次序被加入；它们被放到产生最好目标值的组里。

这种贪婪启发式规则简化了算子：每个排列编码一个有效的分割，所以我们可以使用和货郎担问题相同的算子。毫无疑问，用“贪婪解码”方法比基于其他组–数和分隔点排列[211]解码器的演化程序的性能要好得多。

最近，Falkenauer[109]建议了所谓的分组遗传算法(Grouping Genetic Algorithm, GGA)以处理各种分组（分割）问题。致力于设计适当的染色体表达来获取问题的编码。在这种方法里，一个染色体由两部分组成：一个对象部分(object part)和一个组部分(group part)。对象部分使用组–数编码（在本节前面讨论过）：它是由整数的 n–串组成：

$$(i_1, \ldots, i_n)$$

其中，第 j 个整数 $i_j \in \{1, \ldots, k\}$ 表示分配给对象 j 的组数。染色体的组部分是可变长的且只表示组。例如，下面的染色体：

(1 2 1 3 3 3 1 1 2 : 1 2 3)

按照下面的方式被解释。染色体的第二部分表示有 3 组（指定为 1、2、3）。染色体的第一部分允许对分配的解释：组 1 包括对象{1, 3, 7, 8}; 组 2 有对象{2, 9}; 组 3 有对象{4, 5, 6}。注意在上述表达式中，我们可以用数字“5”替换数字“3”（当然，要在两个部分），而分配的含义保持不变。

这样的表达的关键概念是使遗传算子工作于染色体的组部分上（即第二部分），而染色体的第一部分只是用来对分配进行识别。

例如，对装箱问题（即将 n 个对象装入固定容积的最小数目的箱里），建议的装箱杂交算子 BPCX 这样工作。假定两个亲体染色体为：

(1 2 1 3 3 3 1 1 2 : 1 2 3)

(2 3 3 5 1 4 2 2 6 : 2 3 4 5 6)

染色体的每个组部分的两个杂交位置被随机地选择为：

(1 2 1 3 3 3 1 1 2 : 1|2 3|)

(2 3 3 5 1 4 2 2 6 : 2|3 4|5 6)

然后，第一个亲体的杂交部分的内容在第二个亲体的第一个杂交位置被插入（对另一个后代，第一个亲体和第二个亲体互换角色）：

$$(\ldots :\ \ 2_2\, 2_1\, 3_1\, 3_2\, 4_2\, 5_2\, 6_2)$$

（下标表示亲体）。现在，我们来消除冲突；注意上面列出的箱里的内容如下：

箱 2_2 —— 对象 1，7，8
箱 2_1 —— 对象 2，9
箱 3_1 —— 对象 4，5，6
箱 3_2 —— 对象 2，3
箱 4_2 —— 对象 6
箱 5_2 —— 对象 4
箱 6_2 —— 对象 9

因为有一些共同的对象在“新”箱（来自第一亲体）和“旧”箱中（来自于第二亲体），我们从染色体的第二部分移走这些“旧”箱（引起冲突），它们现在为：

$(\ldots : \ 2_2\, 2_1\, 3_1)$

注意，我们移走了箱 6_2（因为对象 9 已在 2_1 中）、箱 5_2（因为对象 4 已经在 3_1 中）、箱 4_2（因为对象 6 已在 3_1 中）和箱 3_2（因为对象 2 已在 2_1 中）。在此阶段，后代染色体的形状如下：

$(2_2\, 2_1\, ?\, 3_1\, 3_1\, 3_1\, 2_2\, 2_2\, 2_1 : \ 2_2\, 2_1\, 3_1)$

将箱 $2_2, 2_1, 3_1$ 重命名为 1, 2, 3 后，染色体为：

(1 2 ? 3 3 3 1 1 2 ： 1 2 3)

注意由于上述解决冲突的方法的缘故，第三个对象没有被安置（这在染色体第一部分用问号“？”标记）。这样，我们可以使用一些启发式的修补算法来得到一个可行个体：Falkenauer[109]使用了最适下降启发式规则（the first fit descending heuristic）来插入丢失的对象。如果在上述染色体的三个箱的任何一个箱中，没有放对象 3 的地方，则有必要产生一个另外的箱：

(1 2 4 3 3 3 1 1 2 ： 1 2 3 4)

注意这样的杂交可以从两个亲体中传送尽可能多的有意义的信息。

在上述方法中，变异算子非常简单和有用。它选择并消除几个箱（随机地）。没有安置的对象按照最适启发规则被重新插入到箱里（随机次序）。

实算的结果是很好的：Falkenauer 用下面的评语总结其论文[109]：

> “我们同样希望做一个令人信服的实例来说明，适当的编码（因此也是遗传算子）对遗传算法成功典范应用的重要性。”

很难不同意此看法。

11.4　在移动式机器人环境里的路径安排

导航是指导移动式机器人(mobile robot)穿越环境时行程的一门科学（或者艺术）。任何导航方案的本质是要求对任何目标无损失或者无破坏地到达目的地。

机器人的路径通常是离线安排的，假定环境是完全已知且静止的；而且机器人能完美地沿着轨迹移动到达目的地。早期的路径计划者都是这样的离线计划者或者只适合于这样的离线计划（如[248, 218, 205]）。但是离线计划的限制导致研究者研究在线计划，

当机器人穿越环境时，在线计划依赖从感知局部环境所获得的知识[222]来处理未知的障碍物。

这里描述的演化程序，即演化导航器(Evolutionary Navigator， EN)，以高可信度和高效率的演化算法简单地统一了离线和在线计划[243,244]。此算法的第一部分（离线计划者）从开始点到目的地对最优全局解进行搜索，而第二部分（在线计划者）通过用优化子旅行替换原始的全局途径中的一个部分处理可能的冲突或者先前未知的物体。重要的一点是EN的两个部分使用相同的演化算法，只是用不同的各种参数值。

在过去五年中，一些研究者已实算了对路径计划问题的演化计算技术。Davidor[69]使用了染色体动态编码和修正的杂交算子来优化一些真实世界过程（包括机器人路径的应用）。在[355]中描述了一个解路径计划问题的遗传算法，在[356]中推出了实时多启发式搜索策略的遗传算法。这两种方法都假定了一个由结点组成的预定义映射。其他的研究者使用了分级系统[414]或者遗传规划方法[176]来解决路径计划问题。我们的方法在演化导航器的意义上是独特的：（1）操作于整个自由空间并不做有关路径的可信结点的任何优先假设，（2）它组合了离线和在线计划算法。

在详细解释此算法前，我们首先解释映射结构。为了支持在整个、连续的自由空间里的路径搜索，顶点图被用来表示环境中的物体。当前，我们限制环境为两维、只有多边形物体，并只是平移式运动机器人。因此，机器人可以被收缩成点，环境中的物体相应地“变大”[248]。假定EN使用的是一个装备有超声传感器的移动式机器人（一个Denning机器人）。已知物体用其顶点的计划表表示（顺时针方式）。在线遭遇的障碍物由几片“墙”模拟，而每片“墙”都是一条直线，用其两个端点的表表达。这种表达与已知物体的表达是一致的，同时它也适应这样的事实：只有部分有关未知障碍物的信息可以从对一特定位置的感知中获得。最后，整个环境被定义成一个矩形区域。

现在定义EN生成路径是很重要的。一个路径由一个或者多个直线线段组成，并带有起始位置、目标位置及定义结点的两个邻接线段的交叉点位置（可能的话）。一个可行的路径由可行结点组成；不可行路径包含至少一个不可行结点。假定有一个路径$p=\langle m_1, m_2, \ldots, m_n \rangle$（$n \geq 2$），其中$m_1$和$m_n$分别表示起始结点和目标结点。如果结点$m_i$ (i=1,…, n−1)由于障碍物不能和下一个结点m_{i+1}相连或它位于一些障碍物里（或者太靠近），则它是不可行的。假设起始结点和目标结点都位于障碍物以外，并且它们靠得不是太近。不过，注意起始点不需要是可行的（即它和下一个结点可能是不可连接的），而目标结点总必须是可行的。还要注意不同的路径可能有不同的结点数。

现在我们浏览一下EN的过程（图11.1）。

EN首先读取地图并获得任务的起始点和目标点的位置。然后，离线演化算法(oFf–line Evolutionary alGorithm ，FEG)产生一个次优全局路径，即由可行结点(feasible knot points)或者节点(nodes)组成的各直线段相接路径。图11.2展示了由FEG产生的这样一个全局路径（涂黑的圈表示机器人)。

当机器人开始跟从路径并朝目标移动，它从与靠近物体的接近度感知环境，在线演化算法(oN–line Evolutionary alGorithm，NEG）用来处理意外的碰撞和物体产生局部路径。为模拟环境中未知物体的影响，产生的附加数据文件来表示这样的障碍物（如前面

解释的几片“墙”）。我们实算了五个不同的未知物体的设定点；图 11.3（a）～（d）表示了机器人在其中的一个设定点上的行动。

```
procedure Evolutionary Navigator
begin
    begin (off-line planner)
        get map
        obtain the task
        perform planning:
            current path : = FEG(start, goal)
    end (off-line planner)
    if current path is feasible   then
    begin (on-line planner)
    repeat
        move along the current path while sensing the environment
        if too close to any object   then
        begin
            local_start : = current location
            local_goal : = next node on the current path
            if the object is new
                then update the object map
                 else virtually grow the object at the closest spot
            perform planning:
              ocal_path : = NEG(local_start, local_goal)
            update current path
        end
    until (at goal) or (failure condition)
    end (on-line planner)
end
```

图 11.1　演化导航器的结构

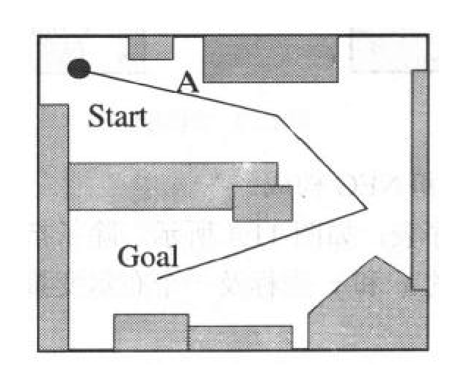


图 11.2　一个环境及全局路径

当机器人的移动太靠近附近的物体“A”的左下角时，NEG 就会当场虚拟“长大”“A”，并生成一个局部路径以操纵离开“A”行进，此路径同样是分段直线段路径。然后机器人跟从当前的路径成功地到达“a”。当机器人从“a”向“b”移动时，他探测到一个未知或者一个新的物体“B”。现在 EN 更新地图，NEG 再一次产生一个带有结点“d”的局部路径（图 11.3（a））。当机器人从 “d”朝“b”移动时，它太靠近物体“B”：因此，另一个局部路径产生，用结点“e”表示（图 11.3（b））。然后机器人从“d”向“e”移动，并最终到达子目标“b”。下一步是从“b”朝目标移动：如图 11.3（c）所示，路径线段太靠近物体“C”的右下角。因此，另一个局部路径产生，用结点“f”表示，然后朝“目标”移动。图 11.3（d） 展示了原始的全局路径和实际旅行的路径。注

意当机器人到达目标，或者报告了一个失败的条件，即当 EN 在某个时间段不能发现一个可行路径（在 NEG 指定的代数里），导航过程终止。

正如我们已经陈述的，EN 以相同的数据编码和同样的计划算法组合了离线和在线行程计划。即 FEG 和 NEG 仅有的不同是它们使用的参数：对 FEG 是群体规模 pop_size_g、代数 T_g、染色体 n_g 的最大长度等。而对 NEG 是 pop_size_l、T_l、n_l 等等。注意 FEG 和 NEG 两者都进行全局计划；尽管 NEG 通常产生一局部路径，但它操作于更新的全局地图。而且，如果在环境中没有已知的初始物体，或者在起始位置和目标位置之间没有初始已知的物体，那么 FEG 将产生恰好是起始位置和目标位置两节点之间的一直线路径。它将只依赖于 NEG 以引导机器人朝着目标移动，同时避免未知的障碍物。

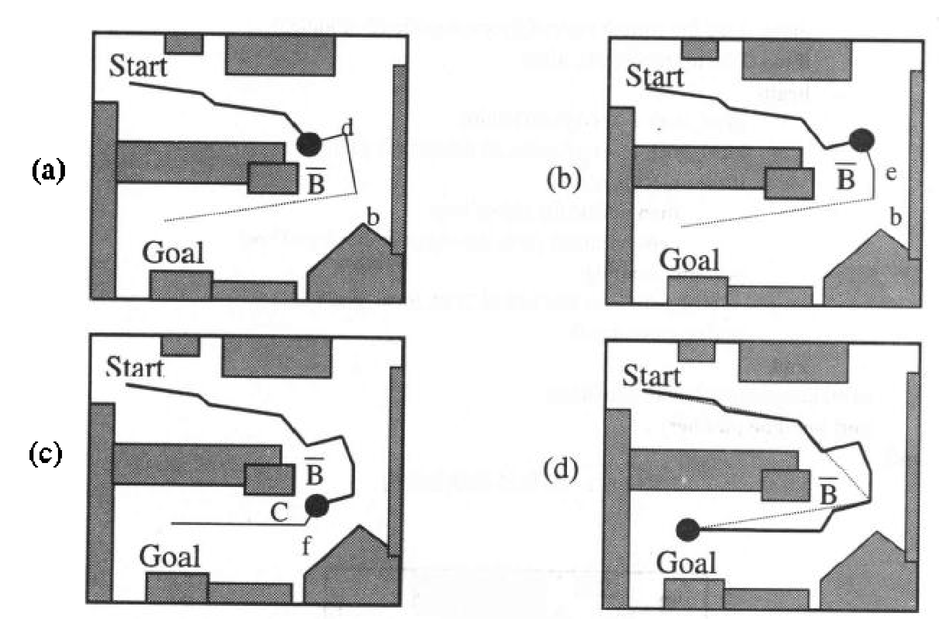


图 11.3 实际行程

下面我们详细讨论 FEG 和 NEG 构成。

染色体为路径节点的次序表，如图 11.4 所示。除了指向下一个节点的指针，每个路径节点由沿着路径中间结点的 x 和 y 坐标及一个布尔变量 b 组成，此布尔变量表示给定节点是可行的还是不可行的。

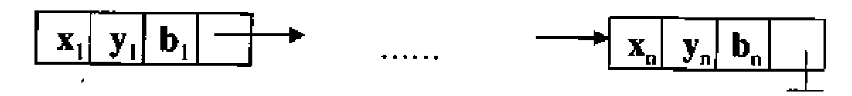


图 11.4 代表一个路径的染色体

染色体长度（在一个染色体中表达的路径节点数）是可变的。在离线计划中，一个染色体的最大长度被设定为代表环境中已知物体的 n_g 个顶点数。不大可能让所有可行路径都要求一较大的中间节点数（即 n_g）：甚至在复杂的环境中，一个可行路径可能十分简单。因此，我们使染色体的长度可变以得体地处理这种情况。

在在线计划中，绕过一个障碍物的局部路径可能只包含少数的节点，因此，参数 n_l（此阶段染色体的最大长度）在此局部搜索开始时是相对地小。但是，如果经过一些代的演化后，演化程序不能找到一个可行路径，染色体的最大长度应该增长：在这种情形下，

十分可能的是可行路径有更复杂的结构。在 EN 系统里，我们假定参数 n_1 是当前代序号 t 的一个函数，更精确地说，$n_1(t)=t$。

染色体的初始群体（对 FEG 为 pop_size_g；对 NEG 为 pop_size_l）是随机产生的。对每个染色体，从范围[2, max(2, n_g)]里（对离线计划者）产生的随机数确定其长度。对这样一个染色体的每个节点，坐标 x 和 y 是随机产生的（当然，坐标的值被限制在环境界限里）。

对每个染色体的每个节点，布尔变量 b 的值是确定的（可行性检验）。如果节点是可行的，其 b 值被设定为 TRUE，否则被设定为 FALSE。一个节点可行性检验的方法（即位置有效性、靠近物体的空隙及连通性）是相对简单的，它基于 Pavlidis[310]描述的算法。

一个染色体 $p=\langle m_1, m_2, \ldots, m_n\rangle$的适应值（总的路径费用）由两个独立的评价函数确定（对可行个体和不可行个体）：

对一可行路径 p:

$$\text{Path_Cost}(p)=w_d\cdot dist(p)+w_s\cdot smooth(p)+w_c\cdot clear(p)$$

其中权值 w_d, w_s和 w_c对一个路径总的费用是归一化的，并且

- $dist(p)=\sum_{i=1}^{n-1} d(m_i, m_{i+1})$，其中 $d(m_i, m_{i+1})$ 是两个结点 m_i和 m_{i+1} 之间的距离；

  即，函数 $dist(p)$返回路径 p 的总长度。

- $smooth(p)=\max_{i=2}^{n-1} s(m_i)$，其中

  $$s(m_i)=\frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}}$$

  即函数 $smooth(p)$返回在一结点上的 p 的最大曲率。

- $clear(p)=\max_{i=1}^{n-1} c_i$，其中

  $$c_i=\begin{cases} d_i-\tau & 若\ d_i\ge\tau \\ \alpha(\tau-d_i) & 若\ d_i<\tau \end{cases}$$

  d_i为路径线段(m_i, m_{i+1}) 和所有已知物体之间的最小距离，τ确定一个安全距离，a 为一个系数；即函数 $clear(p)$返回所有线段 p 和物体之间空隙的最大数值。

对一个不可行路径 p:

$$\text{Path_Cost}(p)=\alpha+\beta+\gamma$$

其中 α为路径 p 和所有物体墙之间的相交点数，β为每个不可行线段相交点的平均数，γ提供当前群体中最差可行路径的费用。因为最后这个变量，使群体中任何可行路径都比任何不可行路径要好（见 15.3 节，C 部分）。

FEG 和 NEG 中包括几个算子（杂交、两个变异、插入、删除、平滑、交换）。我们将依次对它们进行讨论。

杂交 此算子与经典的单点杂交相似，在遗传算法中有广泛的使用。它重组两个亲

体中存在的路径中的“好的”部分，以产生呈现在后代中的有希望的更好路径。两个被选择染色体在某个位置被切割并粘在一起：第一个染色体的第一部分和第二染色体的第二部分，及第二染色体的第一部分和第一染色体的第二部分。但是，两个染色体中的杂交点不是随机选择的：如果染色体中有不可行节点，杂交点落在其中之一之后。

变异_1 该变异用于微调染色体表中节点的坐标值。如果一个染色体一个节点因这种变异被选择，其坐标被修正。例如，坐标 $x \in \langle a, b \rangle$（坐标 y 类似）按照下面的方式改变：

$$x' = \begin{cases} x - \delta(t, x-a) & 若\ r=0 \\ x + \delta(t, b-x) & 若\ r=1 \end{cases}$$

其中 r 为一随机位，函数 $\delta(t, z)$ 返回在$[0, z]$里的一个值，使当 t 增大时（t 为当前演化过程中的代数）$\delta(t, z)$以接近于零的概率增加。对非线性优化（第 7 章），此算子用演化系统中用的此算子被塑造成非均匀变异。该变异用以“平滑”路径的形状。

变异_2 此变异用在要求值有较大变化的情况下（这种情况经常发生在在线计划阶段，当一个障碍物阻塞路径）。如果一个染色体的节点因此种变异被选择，其坐标被修正。例如坐标 $x \in \langle a,b \rangle$（坐标 y 类似）按照下面的方式改变：

$$x' = \begin{cases} x - \Delta(t, x-a) & 若\ r=0 \\ x + \Delta(t, b-x) & 若\ r=1 \end{cases}$$

其中 r 为一随机位，函数$\Delta(t, z)$返回$[0..z]$里的一个值，使当代数 t 增加时，$\Delta(t, z)$以接近于 z 的概率增加。

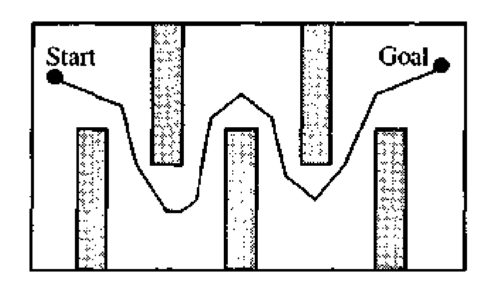
插入 此算子插入一个新的节点到已存在路径中：在两个节点之间每个位置有相同的概率被插入。

删除 此算子删除路径的一个节点；每个节点有相同的概率被删除。

平滑 此算子通过切割急剧的转弯平滑部分的路径。对被选择结点 m_i（有一个较高的曲率），此算子选择两个新结点 k_1 和 k_2（分别从线段 (m_{i-1}, m_i) 和 (m_i, m_{i+1})），插入到它们的路径里，移走 m_i；这样它产生一个新路径 p'：

$$p' = (m_1, \ldots, m_{i-1}, k_1, k_2, m_{i+1}, \ldots, m_n)$$

交换 此算子将选择的染色体分割成两部分（割点是随机确定的），并交换这些部分。



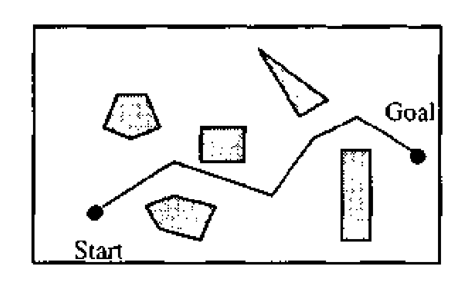
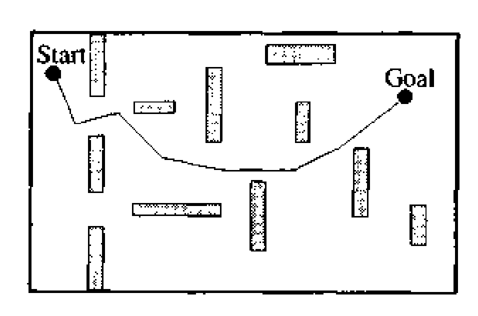


图 11.5 在两个环境下 EN 的结果

基于初始的实算结果，EN 和使用传统方法的导航器比较，已经证明是有效的和有力

的（参见[130]）。对两个不同环境，该系统的当前版本的结果表示在图 11.5 和图 11.6 中。



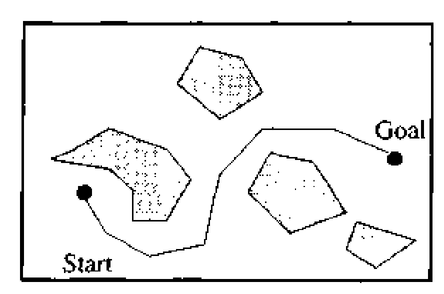


图 11.6　在另两个环境下 EN 的结果

当然，需要通过开发更多的在不同环境下的实算来探究 EN 的潜质，更重要的是通过 EN 在真实的机器人上的实现。同时，这样的演化导航器需要解决几个问题；这包括（1）为更好的实现一个目标，设计对 FEG 和 NEG 更聪明的终止条件（当前的算法终止是：当一个可行路径被找到或者某一固定的代数的耗费）；（2）引入遗传算子适应频率，以替换该系统当前版本中的常数（这种修正应该增强该系统的执行效果并基于简单的观察：不同的算子在演化过程的不同阶段起着不同的作用）；（3）扩展操作于非多边形物体环境中 EN；（4）将搜索当前阶段的知识并入到算子的工作中（例如在不可行结点交叉两个路径可能更有意义）；（5）开拓一些学习机制以便 EN 可以利用过去的经验。

不管其在许多情况下的效率和力度，EN 有一个主要的限制：它假定一个可行并充分好的路径可以从当前最好路径的较小扰动中获得；在穿越的某个阶段，系统的设计不能用另一个（可能更好）全局路径整个地替换当前的全局路径。这样就值得实算其他的解：例如，正在构造中的适应导航器(adaptive navigator，AN)。不像由离线和在线计划组成的 EN，一个适应导航器可以完全是在线计划：它基于最新收集的传感信息不断修改连接当前机器人位置和目标的路径。

11.5　评 述

在最后这一节，我们简要讨论几个相对近的演化技术的应用，这些应用出于各种理由（从构造演化程序的视角看）是有趣的。我们依次讨论它们。

有一些应用（如网络设计问题），其解是一个图。在遗传算法中表示图的问题本身就十分有趣。最近，Palmer 和 Kershenbaum[304]报告了各种方式表示树(tress)的实算。他们发现一个好的表达应有的性质包括：

(1) 完整性：表达所有可能树的能力；

(2) 无偏袒性：所有的树应该用同样的编码数表示；

(3) 固定性：只表示树；

(4) 有效性：树的编码表达和树的表达之间能以一种更普遍的适合于评价适应函数和约束的方式进行简单的转换；

(5) 定位性：树表达式中小的变化带来树的小变化。

当然，树的理想表达应该具有所有上述性质。不过，多数表达并不是这样。在[304]中，作者考虑了几个表达。这包括：

- 特征向量表达，一个树被表达成一个二进制向量，长度等于基图里的边数；
- 前趋表达(predecessors representation)，一个节点被指定为一个根，某个节点的前趋被记录：这里，一个树被编码成一个长度等于节点数的整数向量，
- Prüfer 数表达，一个树被编码成一个 $n-2$ "位" (*digit*)的数（n 为一个树里的节点数），其中每"位"为用一特殊算法确定的一个整数（详情见[304]）。

作者还建议了一个新的表达，它是基于一个简单的观察，即某个节点应该是内部节点而其他的应该是叶节点（leaf nodes）。在这种表达中，染色体保存对每个节点和每个可能的边的一个偏向值（这样，一个树被表达成一个有 $n+\frac{n(n-1)}{2}$ 个数的向量）；图中费用矩阵 C_{ij} 的偏向修正为：

$$C_{ij}' = C_{ij} + P_1(C_{max})\, b_{ij} + P_2(C_{max})(B_i + b_j)$$

其中 P_1 和 P_2 为此方法的参数，C_{max} 为最大连接费用。通过 Prim 的算法，用偏向费用矩阵(biased cost matrix)发现一个最小的覆盖全部节点的生成树，就可找到染色体表达的树。这种表达同样可以编码给定了适当的 b_i 值的任何树。它是一个十分有趣的性质：对此完整的讨论，读者可以参考[304]。

另一篇文献里，Abuali 等人[3]研究了一个新的表达生成树的编码方案（对最小生成树数问题：minimum spanning tree problem）。这种编码方案是基于所谓的确定性代码，它是由 $n-1$ 个整数组成的向量：在确定性代码里第 i 个数，k 对应于从顶点 k 到 $i+1$ 的一条边①。有关这种编码和其他方法的比较及实算结果，见[3]。

Esbensen[101]报告了寻找最优 Steiner 树的遗传算法（它判定是完全 NP 问题）被如下表示：给定一个图和一个指定的顶点子集，任务是找到生成指定顶点的最小费用的子图(subgraph)。作者使用了一个确定性的解码器，它将任何被选择的 Steiner 顶点集解释成一个有效的 Steiner 树。这样，每个染色体为一个二进制串，其中的每一位对应于一个顶点；这样的一个二进制串表示一个 Steiner 顶点的一个子集②。这些顶点的子集和指定的顶点一起构成了解码器的开始点，此解码器（1）构造了这些顶点诱发的子图；（2）计算此子图的最小生成树；（3）通过用原始图中相应的最短路径取代每条边的方法构造另一个子图（来自于此最小生成树）；（4）计算导出的子图是最小生成树；（5）通过重复删除（从最新的最小生成树中）不包含在原始第 1 级的顶点表中的所有顶点，计算 Steiner 树。有关实算结果（包括有 2500 个顶点的图的情况），见[101]。

集合覆盖问题（set covering problem, SCP）是用一列的子集以最小的费用覆盖一个 $n\times k$ 二进制矩阵 $A=(a_{ij})$ 的行；此问题有许多实际的应用（设备的定位、员工日程表等）。对每个列 $1\le j\le k$，关联费用为 c_j；这样 SCP 可以被表达成：

---

① 确定性编码方案可能产生不是生成树的图：在[3]中报告了此工作要使用修补算法。

② Steiner 顶点由附加顶点组成，附加顶点被加入到指定集合里。这样，染色体的长度由整个节点数和指定节点数之差确定。

$$最小化：\sum_{j=1}^{k} c_j x_j$$

其中，x_j 表示一二进制决定变量（当且仅当列 j 选择覆盖，$x_j = 1$），服从约束

$$\sum_{j=1}^{k} a_{ij} x_j \geq 1$$

对所有的 $1 \leq i \leq n$。

Beasley[31]用修正的遗传算法实算了许多 SCP 问题，从 200 行乘 1000 列到 1000 行乘 10 000 列。结果是较好的：算法对较小规模问题产生最优解；对大规模问题产生高质量的解。

注意，SCP 从 GA 的角度有“一个理想的”表达：x_j（$1 \leq j \leq k$）的二进制串表示一个问题的潜在解。不必有另外的代码。评价函数（对可行个体）也是直接的：

$$f(\boldsymbol{x}) = \sum_{j=1}^{k} c_j x_j$$

SCP 中的主要挑战是可行性问题：应用于这样的二进制串的任何算子可能产生违背问题约束的后代（即有些行可能不被覆盖）。Beasley[31]使用了一个修补算法（见第 15 章有关修补算法的一般性讨论）以保持解的可行性。该修补算法负责覆盖未被覆盖的行：对另外列的搜索是基于列费用和将被覆盖而未被覆盖行数的比率。值得注意的是，一旦修补过程完成，严格不可行个体转换为严格可行个体，一个局部优化的步骤就被执行，其中多余的列被移走（即被移走的列没有违反约束）。

Bui 和 Eppley[55]描述了一个混合的遗传算法以求解最大派系问题(maximum clique problem)，并给出了对大到 1500 个顶点和超过 50 万条边测试问题的实算结果。最大派系问题是找到一个给定最大尺寸的图（以顶点数度量）的一个完整的子图（即派系）。该问题的判定方案是一个完全 NP 问题[134]。意料之中，有各种各样已建议的近似方法。注意，任务是找到顶点的子集，因此使用直接的二进制表达是可能的：一个二进制向量

$$\langle b_1, \ldots, b_n \rangle$$

定义了这样的一个子集（对所有的 n 个以随意的次序计划的顶点①，所有的顶点若为 $b_i = 1$，意味着对第 i 个节点的选择）。作者不去设计维持解可行性的复杂的算子（即保证一个后代是一个派系），或者去设计一个修补算法（它可将一任意的解校正成一个派系），而是构造了一个聪明的目标函数，它对非派系和“几乎是”派系作出一个区别；该函数被定义成

$$f(X) = \alpha|X| + \beta \frac{2e(X)}{|X|(|X|-1)}$$

其中α和β为整数（分别称为基数性加权和完整性加权），$e(X)$返回由 X 诱发的子图边数。值得注意的是比率 β/α 是可变的：在算法的开始阶段，它保持较小值（开拓阶段，其中解的基数性被强调），当算法进展时，比率增加（解的完整性被强调）。值得强调的是算法通过并入局部优化方法得以扩展。此启发式方法（1）确定是否移走一特定的顶点

① 见[55]，不过作者实算了不同的预处理步骤；这些步骤按照它们级别的降序排列顶点，或者采用深度优先搜索导出的次序。

（顶点以特殊的次序被考虑）以改进解的适应值（如果是，顶点被移走）；（2）试图增加解的基数性：它确定是否增加一个顶点以增加解的适应值（如果是，顶点被加入）。有关该系统详细的讨论和实算结果，见[55]。

演化技术对集装箱装载(pallet loading) 的有趣应用由 Juliff[215]进行了描述。集装箱装载问题是一个特殊的日程表问题，包含（1）把纸箱装入到集装箱和（2）将集装箱堆积到货车上。要求是十分复杂的，因为要维持交付货中的平衡（所有时间）和装载和卸载纸箱的最大效率。与其他日程表问题（见 11.1 节）一样，可以用直接和间接表达构造此系统；对后一种系统，一个时间表（对此情况是装载构造器）将完成这一工作。但是，这些方法都有一些缺点：高度与问题有关的算子（直接表达）或者受限制的搜索（间接表达）。通常表示一单个性质和其他性质的染色体不能显式地表达（这些附加性质只是合并在装载构造器里），所以遗传搜索不能完全被引导。

Juliff 开发了几个基于间接表达式和智能装载构造器的集装箱装载系统[215]；其中之一合并了多染色体编码以处理问题的各种性质（实际上使用了三个染色体）；意料之中的是，多染色体系统比其他单染色体系统执行的要好。

最近报告了许多演化技术对管理科学中各种问题有趣的应用（包括日程表和时间表问题）。Nissen 新近的综述对这些应用提供了一个完整的参考文献[297]。

演化技术在工业工程中有着越来越高的兴趣；许多文章论述了在此领域的特殊问题及对这些特殊解的描述。有关更多的信息，可见Computers and Industrial Engineering Journal[135]，这是论述遗传算法和工业工程的专门期刊。

让我们用一个总的评述来总结本章：演化技术对组合优化问题越来越多样化的应用（包括 TSP，见第 10 章）引入了一些启发式局部搜索算法以改进它们的性能（对局部搜索技术最新的全面综述，见[2]）。这是作为算法的单个步骤来完成的，或者这样的算法被当作聪明（智能的）变异被采用。Yagiura 和 Ibaraki 在他们最近的研究[411]报告中指出：这种遗传局部搜索算法和其他技术相比，是一更强有力的技术（随机多起点局部搜索或者纯遗传算法）。

这些结果还强调了变异算子的重要性，它不只是背景算子，而且它是任何演化系统的主要组件。这和最近 Jones[213]的结果是一致的，他实算了宏变异并指出在二进制代码中它们的重要性（相对于杂交算子）。无论如何，看起来编码个体所使用的字母的基数性越高，变异算子的角色越重要。对其中个体表达为浮点向量、整数、矩阵、有限状态机等算法而言，就是这种情况。在遗传规划技术（见第 13 章）中，我们并不惊讶地看到一个新的趋势，即强调各种变异算子的角色；同时允许在遗传规划技术中群体规模有显著的减小，因此，更进一步增加了它们的效率。

第12章 机器学习

机器学习主要致力于用输入信息来建立能构造新知识或者改进已获得知识的计算机程序。这种研究多数使用启发式方法来学习，而不是算法式方法。近年来，最活跃的研究领域[284]一直是符号经验学习(symbolic empirical learning，SEL)。该领域主要关心构造、修正一般性符号描述，其结构是未知的优先级(priori)。SEL 中共同的话题是从概念实例开发概念描述[234,284]。特别是，基于属性空间中的问题有着实际的重要性：在许多这样的域里，一方面，提出一套例子事件集相对容易，而另一方面公式化表述假说又十分困难。实现这类监控学习的系统的目标是：

给定例子事件初始集及在概念上它们的成员资格、产生存在于输入集中的概念的分级规则。

取决于输出语言，我们可以将自动知识获取的所有方法分成两类：符号的和非符号的。非符号系统不能清楚地表达知识。例如，在统计模型里，知识是作为一例子集连同一些对应的统计来表达的；在连接模型里(connectionist model)，知识是分布在网络连接之间的[335]。而符号系统在一个高水平描述性语言里产生并保证清楚的知识。这类系统最知名的例子是 AQ 和 ID 系列[81, 281,314]。

本章描述了两个基于遗传的机器学习方法并讨论了由 Janikow 建议的[200]一个演化程序 GIL（遗传归纳学习 — Genetic Inductive Learning）。前两个方法是介于符号和非符号系统之间的：它们在某种程度上使用了高水平的描述性语言，而使用的算子在此语言里是未定义的，并在非符号水平上进行操作。甚至在最近一些面向问题的表达式里见参考文献[149, 363, 340, 91]，算子仍在保守的传统的亚符号水平上操作。第三个系统 GIL 是一个适合于“从例子中学习”的演化程序；该系统合并了与问题有关的知识到数据编码结构和算子中。

下面考虑一个用来贯穿本章的例子。

例 12.1

此例子取自 Emerald 的机器人世界（见[217]和[407]）。每个机器人被六个属性值描述；属性(attribute)及其域(domain)为：

Attributes:	Values of Attributes	属性	属性值
Head_Shape	**R**ound, **S**quare, **O**ctagon	头部形状	圆形、方形、八角形
Body_Shape	**R**ound, **S**quare, **O**ctagon	身体形状	圆形、方形、八角形
Is_Smiling	**Y**es, **N**o	是否微笑	是、否
Holding	**S**word, **B**alloon, **F**lag	手持物	剑、气球、旗子
Jacket_Color	**R**ed, **Y**ellow, **G**reen, **B**lue	夹克颜色	红色、黄色、绿色、蓝色
Has_Tie	**Y**es, **N**o	是否系领带	是、否

黑体字母用来标识属性及它们的值，例如 $J = Y$ 意味着 “Jacket_Color is Yellow”（夹克

的颜色是黄色的）。这些概念描述的例子（其中每个概念 C_i 借助于它们的六个属性和它们的值来描述）为：

C_1 Head is round and jacket is red, or head is square and is holding a balloon

C_2 Smiling and holding balloon, or head is round

C_3 Smiling and not holding sword

C_4 Jacket is red and is wearing no tie, or head is round and is smiling

C_5 Smiling and holding balloon or sword

属性有三种类型：额定的（它们的域是值的集合），线性的（它们的域是线性排列的）和结构的（它们的域是部分排列的）。事件表达了不同的决策类别：从一特殊类的事件构成了其肯定的例子，所有其他的事件构成否定的例子。学习例子以事件形式给出，每个事件是属性值的向量。

概念描述在 VL_1（变量值逻辑系统 — Variable Valued Logic System 的简化版）里被表达[281]， VL_1 是一个被广泛采用的表示在属性空间里操作的任何程序的输入事件的语言。

概念 C 的描述是复合式(complexes)的析取(disjunction)：

$$c_1 \vee \ldots \vee c_k \Rightarrow C$$

每个复合式（c_i）表达一个选择器的合取(conjunction)，有属性关系值设定的选择器是三个一组（如 $\langle J = R\rangle$ 表示“Jacket_Color is Red”）。

上述 C_1～C_5 的概念可以表达成 ：

$$\langle S = R\rangle \wedge \langle J = R\rangle \vee \langle S = S\rangle \wedge \langle H = B\rangle \Rightarrow C_1$$

$$\langle I = Y\rangle \wedge \langle H = B\rangle \vee \langle S = R\rangle \Rightarrow C_2$$

$$\langle I = Y\rangle \wedge \langle H \neq S\rangle \Rightarrow C_3$$

$$\langle J = R\rangle \wedge \langle T \neq Y\rangle \vee \langle S = R\rangle \wedge \langle I = Y\rangle \Rightarrow C_4$$

$$\langle I = Y\rangle \wedge \langle H = \{B,S\}\rangle \Rightarrow C_5$$

注意， 选择器 $\langle T \neq Y\rangle$可以解释成 $\langle T = N\rangle$，因为属性 T 为（额定的）布尔属性，选择器 $\langle H = \{B, S\}\rangle$可以被解释成“Holding Balloon or Sword'”（内含析取）。

难题是如何构造学习概念的系统，即确定所有涉及肯定例子及非否定例子的决策规则。我们可以评价并比较对机器人所有描述系统的错误率和产生规则的复杂性。系统应能预测先前未见例子的分级、或者对部分指定描述的分级（可能不止一个）。

在过去 20 年里，应用演化规划技术到机器学习里的兴趣日益升高（GBML 系统，代表基于遗传的机器学习系统 — genetics based machine learning systems）[88]。这是由于其富有吸引力的思想：表达知识的染色体被遗传算子作为数据处理，同时，被作为执行一些任务的可执行代码。但是，虽然早期的应用是部分成功的（如[323]、[340]、[341]），但也遇到许多问题[86]。通常在 GA 界，有两个论述此问题相互竞争的方法。如 De Jong [86] 所述：

“对任何读过 Holland 的著作[191]的人来说，一个自然的处置方法是将整个规则

表达成一个串（一个个体），维持候选规则集的群体，并使用选择和遗传算子产生新的规则集后代。历史上，这种方法被匹兹堡大学(University of Pittsburgh)的 De Jong 及其学生们采用（例如，可见 Smith 的[363]、[364]），被称为 'Pitt 法'。但在同一时期，Holland 开发了一个认知模型（分级系统），其中，群体的成员是个体规则，并且规则集是由整个群体表达的（例如，见 Holland 和 Reitman[189]; Booker[45]）。这很快变成知名的 'Michigan 法'，并开始了有关这两种方法优缺点的友善的但是争论激烈的一系列讨论。"

我们相信，基于演化规划技术的第三种方法应该是最富有成果的一个。通常，并入与问题有关的知识的思想照例是通过（1）适当的数据编码；（2）与问题有关的"遗传"算子的精心设计来完成的，系统的精度和性能是这种设计的收益。不过 Pitt 法更接近于我们演化规划的思想，因为它维持对问题完整解的群体（规则集），而 Michigan 法（分级系统），通过捆绑、斗链式算法(bucket brigade algorithm)及在修正规则中的遗传组件构造了一个和演化规划技术十分不同的方法。另外，Pitt 法的实现尽管以一高水平的描述性语言表达一个染色体，却并不使用任何学习方法来修正它们的算子。这就是 Pitt 法和演化程序方法的根本不同。

本章将讨论分级系统（Michigan 法，第 12.1 节）、Pitt 法（第 12.2 节）和基于属性实例针对决策规则的归纳学习的 Janikow [200]的演化程序（第 12.3 节）①。

12.1 Michigan 法

分级系统是一类具有并行处理规则、新规则的适应代及已有规则的效率测试遗传机制的基于规则的系统。分级系统提供一个框架，其中被编码成位串的规则的群体的演化是以间歇地从环境给定刺激和强制为基础的。系统"学习"当刺激存在时哪种反应是适当的。分级系统中的规则构成了随时间进化的个体的群体。

一个分级系统（见图 12.1）由下列组件构成：

- 探测器和效应器(detector and effector)
- 消息(message)系统（输入、输出和内部消息表）
- 规则(rule)系统（分级器(classifier)群体）
- 信誉(credit)系统的分配（斗链式算法）
- 遗传步骤（分级器的再生）。

环境发送一个消息（在一个板(board)上移动、新事件的一个例子等），它被分级系统的探测器接受，并被放到输入消息表中。探测器将消息解码成一个或者更多（解码后）的消息并把它们放到（内部的）消息表中。消息激活分级器；强的、激活的分级器将消息放到消息表中。这些新消息可以激活其他分级器或者它们发送一些消息到输出表中。在后一种情况里，分级系统的效应器将这些消息编码成输出消息（在一个板上移动、决

① 有关分级系统更多的信息见[107]；及最近的杂志 *Evolutionary Computation* 中的专题。

策等），它被返回到环境里。环境评价系统的行为（环境反馈——environment feedback），斗链式算法更新分级器的强度。

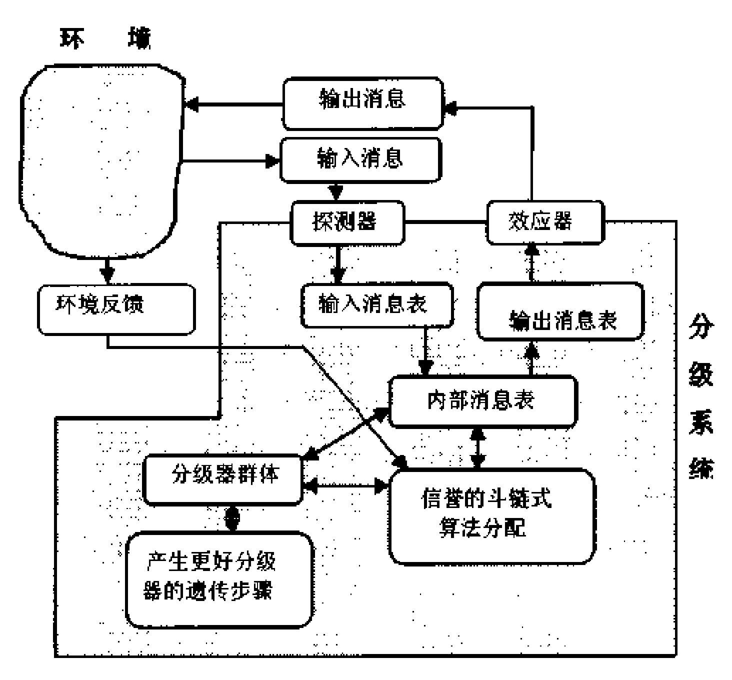


图 12.1 一个分级系统及其环境

现在我们用 Emerald 机器人世界作为例子来更详细地讨论该系统的行为。我们首先提供一些基本的预备性知识。单个的分级器是由两部分组成：（1）条件部分；（2）消息部分。条件部分是一个有限长的一些字母串；这些字母包括“通配”符“*”。消息部分是一个有限长的使用同样字母表的字母串；但是它不包含“通配”符。每个分级器有其自己的强度。强度对命令过程(bidding process)是重要的，其中分级器竞争着传递它们的消息，我们将在此节的后面讨论它们。现在，我们回到 Emerald 的机器人世界。我们可以将一个决策规则表达成一个或多个分级器。每个分级器的格式是：

$$(p_1, p_2, p_3, p_4, p_5, p_6): d$$

其中 p_i 表示上面描述过的第 i（$1 \le i \le 6$）个属性值（例如 $p_4 \in \{S, B, F, *\}$(Holding)）和 $d \in \{C_1, C_2, C_3, C_4, C_5\}$的值。

例如，分级器

$$(R***R*) : C_1$$

表示：“If head is round and jacket is red then concept C_1”（“如果头是圆的且夹克是红的，那么就是概念 C_1”）。

本章中给出的五个概念的分级器集是：

$$(R***R*) : \quad C_1$$
$$(S**B**) : \quad C_1$$
$$(**YB**) : \quad C_2$$
$$(R*****) : \quad C_2$$
$$(**YB**) : \quad C_3$$

$$(**YF**): \quad C_3$$
$$(****RN): \quad C_4$$
$$(R*Y***): \quad C_4$$
$$(**YB**): \quad C_5$$
$$(**YS**): \quad C_5$$

为简化例子，我们假定系统学习一个单个概念 C_1；任何系统可以容易地一般化为处理多概念的系统。在这种情况下，每个分级器的格式是：

$$(p_1, p_2, p_3, p_4, p_5, p_6): \ d$$

其中 $d=1$（概念 C_1 的成员） 或者 $d=0$（其他情况）。

假定在学习过程的某个阶段，在系统中分级器 q 有一小的（随机的）群体（每个分级器和其强度 s 一起被给出）：

$$q_1 = (***SR*) \quad : \quad 1, \quad s_1 = 12.3$$
$$q_2 = (**Y**N) \quad : \quad 0, \quad s_2 = 10.1$$
$$q_3 = (SR****) \quad : \quad 1, \quad s_3 = 8.7$$
$$q_4 = (*O****) \quad : \quad 0, \quad s_4 = 2.3$$

进一步假定来自环境的输入消息（一个新的例子事件）：

$$(RRYSRN)$$

对一单个机器人的描述是圆头(R)、圆体(R)、微笑(Y)、持剑(S)、红色夹克(R)及不带领带(N)。很明显，机器人构造了概念 C_1 的肯定例子，缘于其圆头和红夹克。

三个分级器被此消息激活：q_1，q_2 和 q_4。这些分级器投标：每个标价（bid_i）正比于分级器的强度（$bid_i = b \cdot s_i$）。最强的分级器 q_1 获胜并传递其消息：因为消息被解码成正确的分级，分级器获得一个奖赏 $r>0$；分级器的强度变成

$$s_1 := s_1 - bid_1 + r$$

如果消息被解码成一个错误的回答，“奖赏” r 是负值。对系数 $b = 0.2$ 和 $r = 4.0$，分级器 q_1 的新强度为 $s_1 = 12.3 - 2.46 + 4.0 = 13.84$。

分级系统的参数之一是 GA 的周期 t_{ga}，它规定 GA 调用之间的时间步骤数（即上述循环数）。当然，t_{ga} 可以是一个常数，也可以是随机产生的（平均等于 t_{ga}），或者它甚至不必被指定，是否调用 GA 可以在系统执行基础上作出。这里，我们假定时间已决定应用 GA 到分级器上。

我们将分级器的强度看作是它们的适应值——可以使用赌盘选择（第 2 章）执行选择过程。但是在分级系统中，感兴趣的不再是最强（最适）的分级器，而是执行分级任务的分级器群体。这隐含着我们不是产生整个群体；而是应该更小心选择个体进行替换。通常使用拥挤因子模型（crowding factor model，第 4 章），由于它替换相似的群体成员。

所使用的算子还是变异和杂交。不过有必要作一些修正。这里我们考虑第一属性 *Head_Shape*——其域为集合 $\{R, S, O, *\}$。于是当变异被调用时，我们改变被变异特征成其他三个特征之一（以相等的概率）：

$$R \to \{S, O, *\}$$
$$S \to \{R, O, *\}$$
$$O \to \{R, S, *\}$$
$$* \to \{R, S, O\}$$

后代的强度通常和其亲体相同。

杂交不要求任何修正。我们利用所有分级器都等长这一优点：对两个被选择亲体进行杂交，譬如为 q_1 和 q_2：

$(***|SR*)$: 1

$(**Y|**N)$: 0

我们产生一随机杂交位置点（譬如设为在标记的第三特征之后杂交），则后代为

$(*****N)$: 0

$(**YSR*)$: 1

后代的强度是亲体强度的（可能是加权的）平均值。

现在，分级系统准备继续其学习过程，并开始另一个 t_{ga} 步骤的循环，接受更进一步正和反的事件作为例子，修正分级器的强度。我们希望最终分级器的群体收敛于一些强的个体，例如

$(R***R*)$: 1

$(S**B**)$: 1

$(O*****)$: 0

$(***SY*)$: 0

上面讨论的例子是非常简单的：我们的目的主要是用 Emerald 的机器人世界的一个学习范例来解释分级系统的主要组件。但是注意，我们使用了最简单的投标(bidding)系统（例如我们没有使用有效标价变量 $e_bid = bid + N(0, \sigma_{bid})$，引入一个标准偏差为 σ_{bid}、期望值为零的一些随机噪声），我们也没有使用任何课税(taxation)系统（通常每个分级器被课税以阻止群体朝多产规则倾斜），而且在每步我们只选择单个的获胜者（最强分级器）——通常，不止一个分级器获胜而在消息表中放置其消息。而且，斗链式算法并不是在每个时间步骤都有奖励可得的意义上进行的（对每个提供的例子）。

例子之间没有关系，没有必要跟踪奖励链为消息激活当前（获胜的）分级器分配奖励分。对一些问题，像计划问题(planning problem)，消息部分的长度和条件部分的长度是相同的。对这种情况，一个激活的分级器（旧的获胜者）将放置其消息到（内部的）消息表中，它随后可能激活一些其他的分级器（新的获胜者）。那么，旧的获胜者的强度被一个奖励增强 — 来自新获胜者的偿付。

有关分级系统不同版本的历史透视和完整讨论，读者可参考[154]。

12.2 Pitt 法

Michigan 法可以被理解成一个认知的计算模型：认知实体的知识被表达成一个规则

的集成，它随着时间而修正。我们可以借助于它与环境的相关作用来评价整个认知单元：对单个规则评价（即一单个个体）是没有意义的。

另外，Pitt 法采用的观点是：群体中的每个个体表达整个规则集（分离的认知实体）。这些个体相互竞争，弱的个体死亡，强的个体生存并繁殖：这是通过与它们适应值成正比的杂交和变异的自然淘汰实现的。总之，Pitt 法应用遗传算法到学习问题上。这样做，Pitt 法避免了棘手的信誉赋值问题，对此，启发式方法（如斗链式算法）可以为生成的（应有的或不应有的）行为协作规则集分配正的或负的信誉分。

但是这也带来一些争论。第一个是表达，是否应该用定长二进制向量（带有一个固定的域格式）来表达规则集？这样一种表达对生成新规则是理想的，因此为达目的可以使用经典的杂交和变异。但这似乎太受限制了，而且只适用于对低灵敏度水平下工作的系统。表达属性值的染色体中的基因应该是怎么样的（即基因数等于属性数）？这种决策（如果不被特殊算子支持）似乎不利于工作：如果某域的基数性是大的，系统过早收敛的可能性就会很高[86]，即使是用比平常高得多的变异率。似乎某种内部的表达编码必须在染色体单元之间提供标记，并配合对染色体表达很敏感的算子。这样的方法在[364]和[340]中已实现并被讨论。

Smith[363]更进一步对变长个体进行了实算。他综合了先前仅对定长串 GA 有效的许多结果应用于变长串的 GA 中。

然而，似乎需要一些更大胆的决策致力于表达和算子复杂问题。直到最近[24]，研究者才认识到这种决策是必需的：

> “对此问题的一个解决方案是选择更适合于‘自然表达’的不同遗传算子。没有任何知识是专供传统的基于串的遗传算子用的。GA 的数学分析显示内部表达鼓励有用的基因块冒出，因此能和其他基因块组合以产生改进时，这种表达工作得最好。串表达只是达到此目的的许多方法之一。”

在下一节，我们将推出一个演化程序（基于 Pitt 法），它就是这样做的：表达是丰富且自然的，伴有特殊的、对表达敏感的算子，这些算子直接来自于学习方法论。

12.3　一个演化程序：GIL 系统

已完成的演化程序 GIL[200]使遗传算法（Pitt 法）更靠近符号的水平 — 主要通过定义操作于问题水平特殊算子。GIL 的基本组件和任何其他演化程序一样，是数据编码和遗传算子。系统只为学习单个概念而设计。但是，它可以通过引入多群体很容易地被扩展到多概念环境中。

12.3.1　数据编码

一个染色体表示一个正在学习的概念描述的解：假定任何事件不被属于这种概念的负值描述所覆盖（不属于此概念）。每个染色体是一些复合式的集合（析取）。染色体中复合式的数目可以改变 — 假定所有染色体都是等长的（和 GA 维持一定长串群体一

样），这种假定至少可以说是人工的，所以它和演化规划技术相反。这种决策对 GA 界不是新的：如先前部分描述的，Smith[363]扩展了许多遗传算法的正常结果到变长串中，并实现了一个维持这样（可变长）串的群体的系统。

每个复合式，如 VL_1 中定义的一样，是对应于不同属性的一些选择器的合取。每个选择器本身又含其属性的域的析取。例如，下面是一个染色体（概念 C_1 的描述）：

$$(\langle S=R\rangle \wedge \langle J=R\rangle) \vee (\langle S=S\rangle \wedge \langle H=B\rangle)$$

主要出于效率的原因，对用在染色体的内部表达采用二进制表达的选择器。在位置 i 的二进制数 1 隐含在此选择器中包含第 i 个域值。这就意味着一个属性的域的大小和对应于此属性的二进制子串的长度相等。注意对某个选择器所有 1 的集合等价于此属性中通配符数。这样，描述概念 C_1 的染色体在符号表示上和分级系统中使用的相似，被表达成：

$$\langle R***R*\vee S**B**\rangle$$

而在 GIL 系统中的内部表达是：

$$\langle 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11\rangle$$

其中竖线分隔选择器。注意这样的表达得体地处理了内部析取，如概念 C_5

$$\langle I=Y\rangle \wedge \langle H=\{B,S\}\rangle$$

可以被表达成

$$\langle 111|111|10|110|1111|11\rangle$$

12.3.2 遗传算子

GIL 系统的算子是 Michalski[280]给出的归纳学习方法论上制作的：该方法描述了各种归纳算子，它们构成了归纳推论的过程。它们包括：条件降落(condition dropping)，即从概念描述中降落一个选择器，加入可选规则(adding alternative rule)及降落一个规则(dropping a rule)，扩展基准(extending reference) — 扩展一个内部析取，闭合间隔(closing interval) — 对线性域填入两个存在的值之间失去的值，及爬升归纳(climbing generalization) — 对结构域，爬升归纳树。

GIL 系统分别在三种抽象水平上定义了归纳算子：染色体水平、复合式水平和选择器水平。我们将依次讨论它们。

1. 染色体水平：算子作用于整个染色体

- *RuleExchange*：此算子和经典的 GA 杂交相似，它在两个亲体染色体之间交换被选择的复合式。例如，两个亲体：

  $$\langle 100|111|11|111|1000|11 \vee 010|111|11|010|1111|11\rangle$$
  $$\langle 111|001|01|111|1111|01 \vee 110|100|10|111|0010|01\rangle$$

  可以产生下面的后代：

  $$\langle 100|111|11|111|1000|11 \vee 111|001|01|111|1111|01\rangle$$
  $$\langle 010|111|11|010|1111|11 \vee 110|100|10|111|0010|01\rangle$$

- *RuleCopy*：此算子和 *RuleExchange* 相似，但它从一个亲体中拷贝随机复合式到另

一个。例如，两个亲体：

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 〉

〈 111|001|01|111|1111|01 ∨ 110|100|10|111|0010|01 〉

可以产生下面的后代：

〈 100|111|11|111|1000|11 ∨ 111|001|01|111|1111|01 ∨ 110|100|10|111|0010|01〉

〈 010|111|11|010|1111|11 〉。

• *NewPEvent*：此一元算子合并一个正事件的描述到被选择染色体里。如对一个亲体

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 〉

和一个未覆盖事件

(100|010|10|010|0010|01)

产生下面的后代：

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨100|010|10|010|0010|01 〉

• *RuleGeneralization*：此一元算子归纳复合式的一个随机子集。例如对一个亲体

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨ 100|010|10|010|0010|01 〉

第二和第三个复合式被选择综合，产生下面的后代：

〈 100|111|11|111|1000|11 ∨ 110|111|11|010|1111|11 〉

• *RuleDrop*：此一元算子抽出复数的一随机子集。例如对一个亲体

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨ 100|010|10|010|0010|01 〉

可以产生下面的后代：

〈 100|111|11|111|1000|11 〉

• *RuleSpecialization*：此一元算子限定化复合式的一随机子集。例如，对一个亲体

〈 100|111|11|111|1000|11 ∨ 010|111|11|010|1111|10 ∨ 111|010|10|010|1111|11 〉

第二和第三个复数被选择限定化，产生下面的后代：

〈 100|111|11|111|1000|11 ∨ 010|010|10|010|1111|10 〉

2. 复合式水平：算子作用于染色体的复合式

• *RuleSplit*：此一元算子作用于单个复合式上，将其分裂成一些复合式。例如一个亲体

〈 100|111|11|111|1000|11 〉

可以产生下面的后代（算子分裂第二个选择器）：

〈 100|011|11|111|1000|11 ∨ 100|100|11|111|1000|11 〉

• *SelectorDrop*：此算子作用于单个复合式并“抽出”单个选择器，即，对被选择的选择器所有的值被串“11...1”替换。例如，一个亲体

〈 100|010|11|111|1000|11 〉

可以产生下面的后代（算子工作于第五个选择器上）：

〈 100|010|11|111|1111|11 〉

- *IntroSelector*：此算子是“加入”一个复合式，即，它消除带有串“11... 1”的选择器。例如，--个亲体

  〈100|010|11|111|1111|11〉

  可以产生下面的后代（算子工作于第五个选择器上）：

  〈100|010|11|111|0001|11〉。

- *NewNEvent*：此一元算子合并一个负事件的描述到被选择染色体中。例如，一个亲体

  〈110|010|11|111|1111|11〉

  和被覆盖负事件

  (100|010|10|010|0100|10)

  将产生下面的后代：

  〈010|010|11|111|1111|11 ∨ 110|010|01|111|1111|11 ∨ 110|010|11|101|1111|11 ∨ 110|010|11|111|1011|11 ∨ 110|010|11|111|1111|01〉

3．选择器水平：算子作用于选择器

- *ReferenceChange*：此算子从染色体中，即从选择器之一的域中加入和移走一单个值（0 或 1）。例如，一个亲体

  〈100|010|11|111|0001|11〉

  可以产生下面的后代（注意第四个选择器的不同）：

  〈100|010|11|110|0001|11〉

- *ReferenceExtension*：此算子通过允许一些附加的值扩展选择器的域。对不同类型的属性（额定的、线性的、结构的），它使用不同的选择值概率。例如，一个亲体

  〈100|010|11|111|1010|11〉

  可以产生下面的后代（算子“靠近”第五个选择器的域）：

  〈100|010|11|111|1110|11〉。

- *ReferenceRestriction*：此算子从一个选择器中移走一些域值。例如，一个亲体

  〈100|010|11|111|1011|11〉

  可以产生下面的后代：

  〈100|010|11|111|1000|11〉

GIL 系统十分复杂，而且要求许多参数（例如应用这些算子的概率）。对此和其他实现问题的讨论，读者可以参考[200]。不过，值得注意的是这些算子有一个给定的先验(priori)概率，于是实际的概率是作为这些概率和另外两个参数的函数来计算的：一个参数（先验的）要求在特殊化和一般化之间的平衡，另一个参数（动态的）度量对当前覆盖的函数。这些思想和前面所讨论的相似（第 8 章最后一节）。

所有染色体的每次迭代都以它们的完整性和一致性（如果需要，也可能是费用）进

行评价，一个新群体是由那些更有可能出现的更好个体组成。然后，算子应用于新群体，循环重复进行。

12.4 比较

最近的出版物[407]提供了对许多学习策略的评价，包括一个分级器系统（CFS）、一个神经网络（BpNet）、一个决定树学习程序（C4.5）及一个规则学习程序（AQ15）的评价。这些系统都使用来自Emerald机器人世界的例子；系统必须学习本章开始给出的五个概念（C_1~C_5），同时只看正和负例子的变化百分数（总共有432个不同的机器人存在，即属性值已有的可能的组合）。系统的比较是通过在识别所有的432个机器人（见过的和未见过的）时的平均误差来进行的；结果（取自[407]）在表12.1中给出。

表12.1 不同系统的误差率一览表

系统	学习概要（正%/负%）				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	22.8%	5.0%	4.8%	1.2%	0.0%
BpNet	9.7%	6.3%	4.7%	7.8%	4.8%
C4.5	9.7%	8.3%	11.3%	2.5%	1.6%
CFS	21.3%	20.3%	21.5%	19.7%	23.0%

表12.2给出了GIL系统在个体概念基础上的识别率（取自[200]）。正如所期望的，演化程序GIL比基于分级器方法的CFS系统执行要好得多；令人惊讶的是GIL同样比其他学习系统执行得好。其优势是对小百分数的见过和未见过例子的情况都是很明显的。

表12.2 演化程序GIL误差率一览表

系统	学习概要（正%/负%）				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
C_1	11.1%	5.3%	0.0%	0.0%	0.0%
C_2	0.0%	0.0%	0.0%	0.0%	0.0%
C_3	0.0%	0.0%	0.0%	0.0%	0.0%
C_4	10.4%	0.0%	0.0%	0.0%	0.0%
C_5	0.0%	0.0%	0.0%	0.0%	0.0%

有关这些系统的比较（如生成规则的复杂性）、GIL系统实现问题及其他实算结果的更进一步的讨论，读者可以参考[200]。

12.5 REGAL

一个有趣的从例子中归纳概念描述的方法最近由Giordana和Saitta[139]进行了报导。开发的系统REGAL以析取的正规形式学习概念描述：

$$c_1 \vee \ldots \vee c_k \Rightarrow C$$

每个复合式（c_i）被表达成一个选择器的合取，它包含内部析取，例如

$$\langle S=R \vee O \rangle \wedge \langle J=R \vee Y \rangle \vee \langle S=S \rangle \wedge \langle H=B \vee F \rangle \Rightarrow C$$

关键的问题还是表达。REGAL 系统以定长二进制串工作，因此映射析取的正规形式表达到相应的串是必要的。这可以通过强加一些限制到公式化复合体中实现，它是通过语言模板(template) Λ定义的 —— 表示最大的复合式公式。然后，通过从 Λ中删除一些字母可以获得其他好的形式的公式；在这种方式里，Λ中的文字(literals)可按相应的串位设定。REGAL 系统使用了 2 点杂交和均匀杂交，以及为手边任务而特别设计的一般化和特殊化杂交。

系统在“一次学习一个析取”模式和“一次学习多个析取”模式上进行了测试，其中使用了共享函数。有关该系统和实算结果更详细的描述，读者可以参考[139]；更进一步的实算在[140]中有描述。

第 13 章　演化规划和遗传规划

在本章，我们将简要地综述两个强有力的演化技术：它们是演化规划（13.1 节）和遗传规划（13.2 节）。这两个技术的开发相隔了四分之一世纪，并针对不同的问题；它们对群体中的个体使用了不同的染色体表达，并将重点放在不同的算子上。还有，从我们的“演化程序”视角来看，它们十分相似：针对特定的任务，使用特殊的数据编码（有限状态机和树结构计算机程序）及特殊的“遗传”算子。另外，这两种方法都必须控制结构的复杂性（对有限状态机或树的一些度量可以被并入到评价函数里）。我们将依次讨论它们。

13.1　演化规划

原始的演化规划(Evolutionary Programming, EP)技术是由 Lawrence Fogel[126]开发的。它们主要是针对人工智能的演化过程，具有预测环境中的变化的开发能力。环境被描述为符号序列（从一限定的字母表中），演化算法假想，作为输出产生一个新符号。输出符号应该最大化偿付函数，此函数度量预测的精度。

例如，考虑一系列事件，以符号 a_1, a_2, ...标记，一个算法应该在先前（已知）符号 a_1, a_2, ..., a_n 的基础上预测下一个（未知的）符号，设为 a_{n+1}，演化规划的思想就是演化这样一个算法。

选择有限状态机(Finite State Machines, FSM)作为个体的染色体表达，毕竟有限状态机提供一种基于对符号解释的有意义的行为表达。图 13.1 给出了对奇偶校验的一个简单的有限状态机转换图的例子。这些有向图包含从一个状态到另一个状态的转换、输入和输出值的每个状态和边的结点（从状态 S_1 指向状态 S_2 边旁边的记号 a/b 表示 a 值的输入，同时机器处于状态 S_1，导致输出 b 和下一个状态 S_2）。

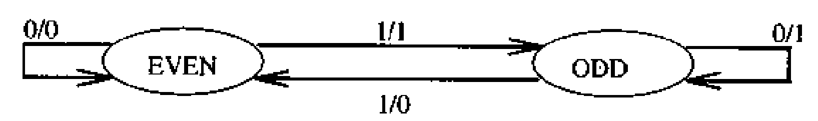


图 13.1　一个用于奇偶检验的有限状态机（FSM）

有两种状态 “EVEN”（奇）和“ODD”（偶）（机器开始于状态“EVEN”），机器可识别二进制串的奇偶。

演化规划技术维持有限状态机的群体，每个这样的个体表示该问题的一个潜在解（即表示一个特殊的行为）。如前所述，每个 FSM 用它的“适应值”度量来评价。这是按照下面的方式完成的：每个 FSM 被放置于环境中，以检查所有先前见过的符号。对每个序列，譬如为 $a_1, a_2, ..., a_i$，它产生一个输出 a'_{i+1}，此输出和下一个观察到的符号 a_{i+1} 进行

比较。例如，如果到目前为止，见过 n 个符号，FSM 做 n 个预测（每个子串 $a_1; a_1; a_2; \ldots$ 等；直到 $a_1, a_2, \ldots, a_n$）；适应函数考虑整个的执行效果，例如对所有 n 个预测精度的加权平均。

就像演化策略一样（8.1 节），演化规划技术首先产生后代并随后为下一代选择个体。每个亲体产生单个后代；因此中间群体的大小被加倍（和 (*pop_size*, *pop_size*)–ES 一样）。后代（新的 FSM）由亲体群体的随机变异产生（见图 13.2）。有五种可能的变异算子：输出符号的改变、状态转换、状态加入、状态删除、初始状态改变（还有对状态最小和最大数目的附加限制）。这些变异以一些概率分布被挑选（概率在演化过程中可以变化）；还有，也可能对单个亲体应用不止一次的变异（对一特定个体，变异数的确定是以某种概率分布作出的）。

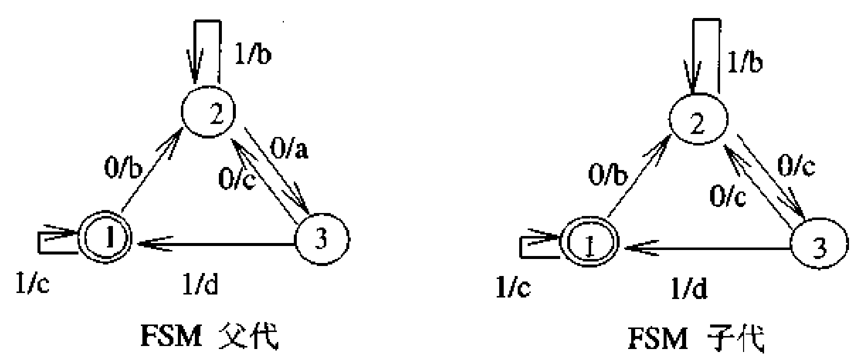


图 13.2 一个 FSM 及其后代。机器开始于状态 1

最好的 *pop_size* 个个体被保留到下一代；即，保留在下一代中的合格个体应该排列在中间群体顶端的 50%。在原始版本[126]中，这一过程在下一个输出符号获得前被迭代执行若干次。一旦得到一个新符号，它就被加入到已知符号表中。整个过程重复进行。

当然，上述过程可以在几个方面进行扩展，如[121]中所述：

> “支付函数可以是任意复杂的，且可以拥有临时组件；对经典的平方误差准则或者任何其他平滑函数，没有其他要求。更进一步，不要求进行向前一步的预测。预测可以在未来的任意长度的时间里完成。可以处理多变量环境，且环境过程不需要是静态的，因为模拟进化将适应转换统计量的变化。”

如 8.2 节所述，演化规划技术可以推广到处理数值优化问题；有关细节见[117]或[121]。有关演化规划技术的其他例子，可以参考[126]（整数序列分级成素数和非素数）、[120]（EP 技术应用于囚犯困境问题），还可参考[123, 124, 378, 254]中对许多其他问题的应用。

13.2 遗传规划

另一个有趣的方法最近由 Koza[228,231]开发。Koza 建议要得到的程序应该在演化过程中自身进化。换句话说，不是单纯地解决一个问题，也不是建立一个演化程序去解决问题，而是我们应该从可能的计算机程序空间来搜索最好的一个（最适的）。Koza 开发的

新方法名为遗传规划(Genetic Programming，GP)，提供这样搜索的一个运行方式。一个可执行计算机程序的群体被产生，个体程序相互竞争，较弱的程序死亡，较强的程序繁殖（杂交、变异）……

对一个特定问题，遗传规划的五个主要步骤是：

(1) 终点的选择；

(2) 函数的选择；

(3) 评价函数的识别；

(4) 系统参数的选择；

(5) 终止条件的选择。

特别值得注意的是，经历演化的结构是一个等级结构计算机程序① 。搜索空间是一个有效程序的超空间，此空间可以被看成是一个根树空间。每个树是由函数和适合特定问题域的终点组成；所有函数和终点集合以一些构成树产生一个解的方式选择优先级。

例如，两个结构 e_1 和 e_2（图 13.3）分别代表表达式 $2x+2.11$ 和 $x\cdot\sin(3.28)$。一个可能的后代 e_3 （经过 e_1 和 e_2 的杂交后）表示 $x\cdot\sin(2x)$。

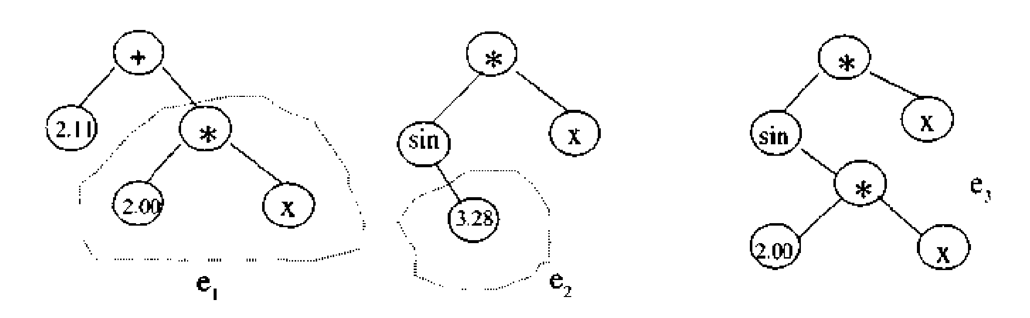


图 13.3 表达 e_3：e_1 和 e_2 的一个后代。虚线围的区域表示杂交交换的位置

初始群体是由这样的树构成；一个（随机）树的构成是直接的。评价函数分配一个适应值来评价树（程序）的执行结果。评价是基于测试例子集的预选择；总之，评价函数返回所有测试例子的正确结果和所得结果之间的距离和。选择是比例选择：每个树在下一代中被选择的概率正比于其适应值。最初的算子是杂交，从两个被选择亲体中产生两个后代。杂交通过交换两个亲体之间的子树产生后代。也有其他算子：变异、排列、编辑和定义基因块操作[228]。例如，一个典型的变异在树里选择一个结点并从被选择结点产生一个新的（随机的）子树。

对一特定问题，除了建立遗传程序的五个主要步骤外，Koza[332]最近考虑了把过程集做为一个新特征加进去的优点。这些过程被称为自动定义功能块(Automatically Defined Functions, ADF）。似乎这对遗传规划技术是极有用的概念，其主要贡献是代码的复用性，ADF 开发并利用手边问题的规律性(regularities)、对称性(symmetrices)、相似性(similarities)、模式(patterns)和模块性(modularities)，而且最终遗传程序可能在其执行的不同阶段调用这些过程。

① Koza 实际上对所有实算采用了 LISP 语言的 S-表达式。不过，近来也有用 C 和其他编程语言实现的 GP。

或许将遗传规划分类成演化规划的另一版本可能是错误的，特别是看作对带有特殊个体染色体表达的遗传规划。事实上，用在计算机程序上的遗传规划算子只有几个方面让人感兴趣。如，算子同样可以被看成是程序，这些程序在系统运行期间将经历分离的进化。另外，一函数集可以由几个执行复杂任务的程序构成；这些程序可以在演化运行期间更进一步进化（如 ADF）。很明显，这在当前演化计算领域的发展中是最激动人心的，而且现在已积累了大量的有意义的实算数据（见[231]、[232]、[225]和[8]）。

第 14 章　演化程序的等级

在本书中，我们讨论了称为演化程序的不同方法，它们可以用在较难的优化问题上，都是基于进化原理的。演化程序大量借用遗传算法的思想。通过使用“自然的”数据编码结构和对问题敏感的“遗传”算子引入与问题有关的知识。遗传算法（GA）和演化程序（EP）之间的主要不同是前者被认为是弱方法，即与问题无关的方法，而后者不是。

弱和强方法之间的区分并不十分明显。不同的演化程序可以从展示问题相关性的不同角度来建立。对一个特定问题 P，通常有可能构造一族演化程序 EP_i，它们中的每一个都可以“解决”该问题（图 14.1）。“解决”一词的意思是“提供一个合理的解”，即一个不必是最优的但是满足问题的约束的可行的解。

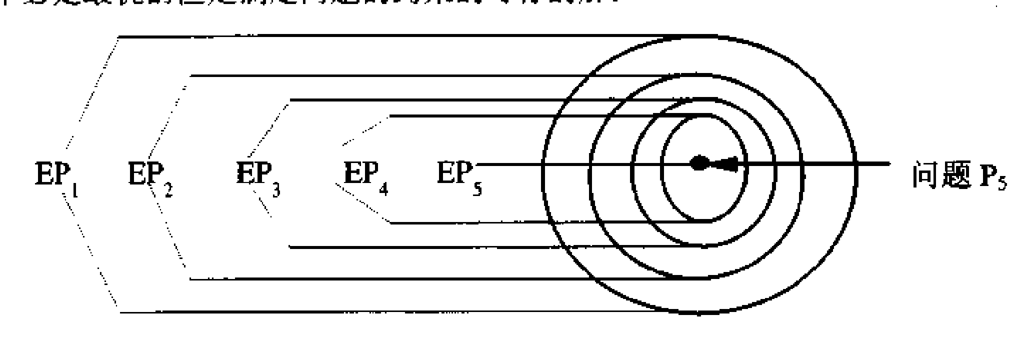


图 14.1　演化程序的等级

演化程序 EP_5（图 14.1）更与问题有关，且只适合问题 P。系统 EP_5 对该问题的任何修正版本都不能很好地工作（如加入一个新的约束或者改变问题的大小）。下一个演化程序 EP_4，可以用在运行一小类问题，这包括问题 P；其他演化程序 EP_3 和 EP_2 可以工作在更大的域，而 EP_1 是与域无关的，而且可以用在任何优化问题上。

我们已经在本书的许多地方看到了这样的等级。现在考虑一个特定的 20×20 的非线性运输问题 P。其中有 400 个变量和 20+20=40 个等式（其中 39 个是相互独立的）。另一个约束要求变量必需取非负值。原则上能构造一个演化程序来解决这一特定问题，设为 EP_5。它可以是带 39 个微调这些约束的罚函数或解码器或修补算法的一个程序。任何对问题大小的变化（从 20×20 到 20×21）或一个出发地到一个目的地运输费用上的变化都将导致 EP_5 系统发生错误。

演化程序 GENETIC-2（第 9 章）可以用于任何运输问题。设该系统为 EP_4。此系统仍属于强方法一类，因为它只能用于非线性运输问题。但是，它比 EP_5 弱，因为它能处理任何运输问题。

另一个演化程序 GENOCOP（第 7 章）EP_3 可用于问题 P 上。它可以优化带线性约束的函数，当然也适合运输问题 P。很明显，EP_3 比起 EP_4 是一个较弱的方法。但它仍然可

以被认为是一个相当强的方法，因为它只能用于带线性约束的数值优化问题。

另一个可以应用于 20× 20 运输问题 P 的演化程序（第 8 章）可以应用于任何带不等式约束（不一定是线性的）的数值优化问题，称之为 EP_2。很明显，问题 P 属于域 EP_2；比起 EP_3，EP_2 也是一个较弱的方法，因为它能处理任何类型的不等式约束（对问题 P，等式可以很容易地用第 7 章所讨论的方法用不等式替换）。

我们同样可以构造一个通用的演化程序 EP_1，它可以是带一标准罚函数集的经典遗传算法：每个罚函数对应于问题的一个约束。系统 EP_1 是与域无关的，它可以处理带任何约束集的任何优化问题。对数值优化问题，约束可以包括非线性等式，它使该系统比只限于不等式的 EP_2 弱。而且 EP_1 也可以应用于其他非数值的问题。这里假定问题 EP_1 总是返回一个可行解。如果初始群体是由可行解和罚函数组成，或者如果修补算法在其搜索空间里保存个体，那么我们可以很容易地加强它。

我们用 $dom(EP_i)$表示一个任意可用演化程序 EP_i 解决的问题集合，即程序返回一个可行解。很明显，

$$dom(EP_5) \subseteq dom(EP_4) \subseteq dom(EP_3) \subseteq dom(EP_2) \subseteq dom(EP_1)$$

然而上述例子是不完全的：它可能产生其他一些适合 EP_i 和 EP_{i+1} 的演化程序，其中 $1 \le i \le 4$。当然，上述等级里有可能存在和其他演化程序重叠的演化程序。例如，我们可以构造带有多项式费用函数的系统来优化运输问题，或者是优化限定在凸搜索空间的问题，或者是带有非线性约束的问题。换句话说，演化程序集合是部分排序的：我们用$\prec$按照下面的意义表示次序关系：如果 $EP_p \prec EP_q$ ，那么演化程序 EP_p 是比 EP_q 弱的方法，即 $dom(EP_q) \subseteq dom(EP_p)$。参考运输问题的例子 P 和一个演化程序的等级 EP_i，得：

$$EP_1 \prec EP_2 \prec EP_3 \prec EP_4 \prec EP_5.$$

假设 $EP_p \prec EP_q$，那么较强的方法 EP_q 比较弱的方法 EP_p 总体上执行得更好。我们对这一假定没有做任何的证明，因为它只是基于大量的实算和简单的直觉，即与问题有关的知识增强其执行效果（时间和精度）的同时限制了其应用广度。我们已经看到了 GENETIC-2 比 GENOCOP 执行得更好，下面将讨论 GENOCOP 如何在一类特殊的问题上比经典的遗传算法执行得好。如果这一假定是真的，GENOCOP 比一个基于处理线性约束问题的演化策略的演化程序给出更好的结果，因为 ES 是比 GENOCOP 更弱的方法。一些其他的研究者支持上述假定：可从 Davis[77]中获得引证：

“在专家系统领域，域的知识能增加优化的执行效果是众所周知的，根据我应用遗传算法到工业问题的经验，这种办法也是适用的。二进制杂交和二进制变异都是与知识无关的算子，因此，如果不把知识引入到我们的遗传算法中，这些算法就很有可能比几乎所有考虑了域知识的优化算法执行得差。”

Goldberg[156, 154]有另外一个看法。他在[156]中认为：

“当然，人们对很窄的一类问题已经开发了非常有效的搜索方法，对连续的、二次优化问题，遗传算法不可能对结合方向或梯度的方法构成威胁。但是这并不是主要的。……结合应用宽度和相对效率（即使不是峰值，*peak*）一起定义

了遗传搜索的主要话题：鲁棒性(*robustness*)。”

我们将这一观察绘在图 14.2 中，其中经典的方法 Q 能很好地解决一个问题 P，对其他问题则不行，而遗传算法却合理地跨过了这一局限（图 14.2 是[154, 156]中给出的相似的图的简化）。

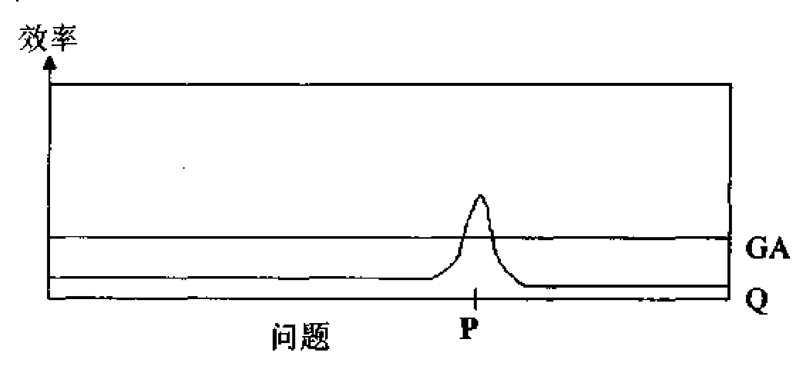


图 14.2　效率/问题图及遗传算法

但是，当非常规的强约束存在时，遗传算法的执行效果却经常被破坏。而提供了一些与问题有关知识的演化程序可能比经典的方法执行得好（图 14.3）。

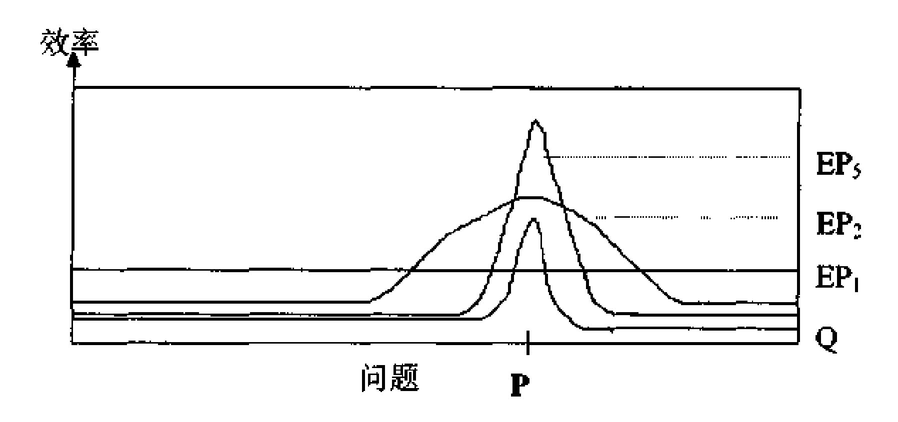


图 14.3　效率/问题图及 EP

应该再次强调，在本书中出现的多数演化程序并没有更多的理论支持。无论是模式定理（如经典的遗传算法），还是收敛定理（如演化策略）都是这样。同样重要的是演化程序通常比其他优化技术要慢。另外，它们的时间复杂性经常是随着问题的尺寸而以线性（或者 $n \log n$）增长，多数其他技术则不是这种情况。最近，Nick Radcliff 对用于任意数据编码的遗传算子进行了形式和性质分析（即重组）[315, 316, 317]，其工作为演化程序提供一些理论证明迈出了一大步。

最近的一系列实算支持上述讨论[261]。强弱演化程序的思想被测试于一特定的非线性运输问题 P 和五个演化程序 EP_i（$i=1,\ldots,5$）。我们将在下面的段落里讨论这些实算。

首先定义一特定的非线性平衡运输问题 P。假定 3 个来源地和 4 个目的地。供货为：

$$source(1) = 10, source(2) = 15, source(3) = 20$$

需求为：

$$dest(1) = 3, dest(2) = 20, dest(3) = 5, dest(4) = 17$$

问题 P 总流量是 45。正如在第 9 章所讨论的，对非线性运输问题的最优解可能既不

包含零值，也不包含整数值，像线性运输问题一样。例如，对一些运输费用函数 f_{ij}，下面的解可能是最优的：

	3.0	**20.0**	**5.0**	**17.0**
10.0	1.34	1.52	0.01	7.13
15.0	1.15	10.39	0.39	3.07
20.0	0.51	8.09	4.60	6.80

对测试问题 *P*，对每个流量 f_{ij} 使用相同的函数 *f*；并用一个费用矩阵来提供流量之间的变化。矩阵所提供的 c_{ij} 用来缩放基本函数形状。

流量 x_{ij} 采用下面的函数 *f*：

$$f(x_{ij})=\begin{cases}0 & \text{若 } x_{ij}=0\\ d+c_{ij}\cdot\sqrt{x_{ij}} & \text{若 } x_{ij}\neq 0\end{cases}$$

其中，i=1,2,3，j= 1,2,3,4，$d=5.0$ 以及

$$\begin{array}{llll} c_{11}=0.0 & c_{12}=21.0 & c_{13}=50.0 & c_{14}=62.0\\ c_{21}=21.0 & c_{22}=0.0 & c_{23}=17.0 & c_{24}=54.0\\ c_{31}=50.0 & c_{32}=17.0 & c_{33}=0.0 & c_{34}=60.0.\end{array}$$

所以问题 *P* 即为求最小：

$$\sum_{i=1}^{3}\sum_{j=1}^{4}f\ (x_{ij})$$

并服从下面的约束：

$$\begin{aligned} &x_{11}+x_{12}+x_{13}+x_{14}=10\\ &x_{21}+x_{22}+x_{23}+x_{24}=15\\ &x_{31}+x_{32}+x_{33}+x_{34}=20\\ &x_{11}+x_{21}+x_{31}=3\\ &x_{12}+x_{22}+x_{32}=20\\ &x_{13}+x_{23}+x_{33}=5\\ &x_{14}+x_{24}+x_{34}=17\end{aligned}$$

我们使用 GAMS（见第 6 章）解决上述问题 *P*。GAMS 的最好解为：

$$\sum_{i=1}^{3}\sum_{j=1}^{4}f\ (x_{ij})=430.64,$$

它是这样得到的：

$$\begin{array}{llll} x_{11}=3.0 & x_{12}=0.0 & x_{13}=0.0 & x_{14}=7.0\\ x_{21}=0.0 & x_{22}=5.0 & x_{23}=0.0 & x_{24}=10.0\\ x_{31}=0.0 & x_{32}=15.0 & x_{33}=5.0 & x_{34}=0.0\end{array}$$

这个结果将作为评价这里的演化程序的参考点，也就是把 GAMS 当作对问题 P 的一个经典的基于梯度的方法，见图 14.3。

为了对演化程序 EP_i（i =1,...,5）进行公正比较，设定群体规模为 70，且所有的实算

取代数为 5000。每个实算重复 20 次；对一特定实算，下面部分引用的评价值为 20 次运行的平均。很重要的一点是这里的演化程序使用了不同的初始化技术，我们将在以后讨论它们。

1. 演化程序 EP_1

在实算中使用的弱演化程序 EP_1 为 GENESIS 1.2ucsd 系统①，该系统由美国加州大学圣地亚哥分校的 Nicol Schraudolph 开发（系统是基于由 John Grefenstette 写的遗传算法包 GENESIS 4.5）。原则上，可以使用这样的遗传工具优化各种问题，而且 $dom(EP_1)$实际上是无限的。

下面我们对给出的测试实例问题 P 练习使用这个演化程序。很明显，如果不把罚函数引入约束，系统将不提供任何有用的解。对 EP_1 运行几次，对十二个变量中的每一个只定义一个域。这里没有更多选择的是选择每个变量的域为从零到对一给定行和列的较小的边界值的和：

$$0.0 \le x_{11} \le 3.0 \quad 0.0 \le x_{12} \le 10.0 \quad 0.0 \le x_{13} \le 5.0 \quad 0.0 \le x_{14} \le 10.0$$
$$0.0 \le x_{21} \le 3.0 \quad 0.0 \le x_{22} \le 15.0 \quad 0.0 \le x_{23} \le 5.0 \quad 0.0 \le x_{24} \le 15.0$$
$$0.0 \le x_{31} \le 3.0 \quad 0.0 \le x_{32} \le 20.0 \quad 0.0 \le x_{33} \le 5.0 \quad 0.0 \le x_{34} \le 17.0$$

很明显，该程序找到的解都不满足问题的约束条件；一个典型的输出为：

$$x_{11} = 2.05 \quad x_{12} = 0.00 \quad x_{13} = 0.00 \quad x_{14} = 0.00$$
$$x_{21} = 0.00 \quad x_{22} = 10.65 \quad x_{23} = 0.00 \quad x_{24} = 0.00$$
$$x_{31} = 0.00 \quad x_{32} = 0.00 \quad x_{33} = 0.00 \quad x_{34} = 0.00$$

正如所预测的，上述非可行解对用户没有任何价值。它可以进一步“改进”：对所有的 $1 \le i \le 3, 1 \le j \le 4$，一个解 $x_{ij} = 0.0$ 得到最优的运输费用是零！

这里显然有必要引入一些约束惩罚。因为演化程序 EP_1 不应该依赖于所解决的问题，这里只是实算了一些标准的罚函数。我们考虑了两个罚函数集。第一个（p_i，中度惩罚）对违反约束施以线性函数惩罚，另一个集合（q_i，高度惩罚）对违反约束施以平方惩罚。对该问题，七个线性等式是：

$$p_1 = c \cdot |x_{11} + x_{12} + x_{13} + x_{14} - 10|$$
$$p_2 = c \cdot |x_{21} + x_{22} + x_{23} + x_{24} - 15|$$
$$p_3 = c \cdot |x_{31} + x_{32} + x_{33} + x_{34} - 20|$$
$$p_4 = c \cdot |x_{11} + x_{21} + x_{31} - 3|$$
$$p_5 = c \cdot |x_{12} + x_{22} + x_{32} - 20|$$
$$p_6 = c \cdot |x_{13} + x_{23} + x_{33} - 5|$$
$$p_7 = c \cdot |x_{14} + x_{24} + x_{34} - 17|$$

且 $q_i = p_i^2/c$ （i=1,...,7）。对所有的实算，c=10.0；对此数，惩罚是由一重要的总费用的百分数组成，总费用如 GAMS 系统所得到的，在 400 左右。实算结果十分有趣。

① 该系统是在动态参数编码选项下运行的（Schraudolph & Belew, 1992）；但是该选项并不改进系统的执行效果，因为精度不是其中考虑的因素。对以后讨论的演化程序 EP_5 也是这样。

下面的点表示一个对带有惩罚 p_i 的实算的典型结果：

$x_{11}=3.00$　$x_{12}=3.77$　$x_{13}=0.00$　$x_{14}=0.00$
$x_{21}=0.00$　$x_{22}=1.23$　$x_{23}=0.00$　$x_{24}=13.77$
$x_{31}=0.00$　$x_{32}=15.00$　$x_{33}=5.00$　$x_{34}=0.00$

上述解只是“典型的”：我们不能得到最好的结果，这意味着很难评价非可行解的好坏程度。为从一非可行解得到一个可行解，可以作一些调整，而且最终的运输费用依赖于这种调整。例如上述解可以校正成下面的可行解：

$x_{11}=3.00$　$x_{12}=3.77$　$x_{13}=0.00$　$x_{14}=3.23$
$x_{21}=0.00$　$x_{22}=1.23$　$x_{23}=0.00$　$x_{24}=13.77$
$x_{31}=0.00$　$x_{32}=15.00$　$x_{33}=5.00$　$x_{34}=0.00$

它得到的总的运输费用为 453.43。当然，一些其他的校正能得到较好或者较差的运输费用。上述校正是手工完成的。它是根据一个简单的观察，第一行和第四列的总和比相应定义的边界值和小 3.23；因此把 3.23 加到 x_{14} 里。

上述例子对非可行解的手工校正产生一个很有价值的值是 453.43。但是应该强调的是只有对低维问题它才是可能的。对一个 20×20 的运输问题是不可行的，找到一个“较好”校正的过程可能和解原始问题一样困难。看起来应该用强的惩罚强制解进入到可行区域。

确实，强惩罚的方法提供了“几乎”可行的解。下面点表示对带有惩罚 q_i 实算的最好结果：

$x_{11}=3.00$　$x_{12}=6.98$　$x_{13}=0.00$　$x_{14}=0.00$
$x_{21}=0.00$　$x_{22}=0.00$　$x_{23}=3.06$　$x_{24}=11.93$
$x_{31}=0.00$　$x_{32}=13.02$　$x_{33}=1.93$　$x_{34}=5.03$

上述解可以很容易地用手工调整的方法转变成一个可行解：

$x_{11}=3.00$　$x_{12}=7.00$　$x_{13}=0.00$　$x_{14}=0.00$
$x_{21}=0.00$　$x_{22}=0.00$　$x_{23}=3.00$　$x_{24}=12.00$
$x_{31}=0.00$　$x_{32}=13.00$　$x_{33}=2.00$　$x_{34}=5.00$

它产生的总运输费用是 502.53。此费用要比用中度惩罚方法所获得的费用 453.43 差。但是应该强调的是在中度惩罚方法里。找到一个“较好”的校正过程对高维问题是十分困难和复杂的。我们可以将这一步考虑成解决一个新的带有修正边界和（此和表示实际的与要求的总和之间的差异）的运输问题的一个过程，其中变量设为δ_{ij}，表示对原始变量 x_{ij} 相应的校正。这样，总的来说强惩罚得到较好的结果。同时，这些结果仍然比从商业软件 GAMS 所获得的结果差（图 14.3 中的系统 Q）。同时应该指出的是“非常强”的惩罚并不改进程序的执行结果。极而言之，如果我们对违反约束的个体分配零适应值，系统经常会驻留在找到的第一个可行解上。

对 EP_1 的实算最终的，也是可以预测的结果是：使用罚函数并不保证获得可行解，而且“好”的修补可能代价昂贵。

2. 演化程序 EP_2

正如第 8 章所讨论的，演化策略假定一个 $q \geq 0$ 的不等式集合作为优化问题的一部分，

$$g_1(x) \geq 0, ..., g_q(x) \geq 0$$

如果在某些迭代过程中，一个后代不满足所有的约束，那么此后代即丧失资格，即它不被放入到新群体中。如果这样的非法后代发生的比率较高，ES 就会通过提供较少向量$\boldsymbol{\sigma}$的组分调整其控制参数。

我们已经使用了 KORR 2.1，由 Hans–Paul Schwefel 和 Frank Hoffmeister 实现的一个 $(\mu+\lambda)$–ES 和 (μ, λ)–ES 演化策略作为我们的下一个演化程序 EP_2。很明显，演化策略可用于参数优化问题，所以 $dom(EP_2) \subseteq dom(EP_1)$ 和 $EP_1 \prec EP_2$。

如前所述，EP_2 只处理不等式约束。因为问题 P 可被重写以消去等式。结果是目标函数只有六个变量：y_1，y_2，y_3，y_4，y_5和 y_6，约束问题 P 如下给定：

$$\begin{aligned}\min \quad & f(y_1)+f(y_2)+f(y_3)+f(10.0-y_1-y_2-y_3)+f(y_4)+f(y_5)+f(y_6)\\ & +f(15.0-y_4-y_5-y_6)+f(3.0-y_1-y_4)+f(20.0-y_2-y_5)\\ & +f(5.0-y_3-y_6)+f(y_1+y_2+y_3+y_4+y_5+y_6-8.0)\end{aligned}$$

其中

$$y_1 = x_{11}, \quad y_2 = x_{12}, \quad y_3 = x_{13}, \quad y_4 = x_{21}, \quad y_5 = x_{22}, \quad y_6 = x_{23}$$

及下面的十八个约束：

$$\begin{aligned}
&g_1: \ y_1 \geq 0 \ （即, x_{11} \geq 0）,\\
&g_2: \ y_2 \geq 0 \ （即, x_{12} \geq 0）,\\
&g_3: \ y_3 \geq 0 \ （即, x_{13} \geq 0）,\\
&g_4: \ y_4 \geq 0 \ （即, x_{21} \geq 0）,\\
&g_5: \ y_5 \geq 0 \ （即, x_{22} \geq 0）,\\
&g_6: \ y_6 \geq 0 \ （即, x_{23} \geq 0）,\\
&g_7: \ 10.0-y_1-y_2-y_3 \geq 0 \ （即, x_{14} \geq 0）,\\
&g_8: \ 15.0-y_4-y_5-y_6 \geq 0 \ （即, x_{24} \geq 0）,\\
&g_9: \ 3.0-y_1-y_4 \geq 0 \ （即, x_{31} \geq 0）,\\
&g_{10}: \ 20.0-y_2-y_5 \geq 0 \ （即, x_{32} \geq 0）,\\
&g_{11}: \ 5.0-y_3-y_6 \geq 0 \ （即, x_{33} \geq 0）,\\
&g_{12}: \ y_1+y_2+y_3+y_4+y_5+y_6-8.0 \geq 0 \ （即, x_{34} \geq 0）,\\
&g_{13}: \ 3.0-y_1 \geq 0 \ （即, x_{11} \leq 3）,\\
&g_{14}: \ 10.0-y_2 \geq 0 \ （即, x_{12} \leq 10）,\\
&g_{15}: \ 5.0-y_3 \geq 0 \ （即, x_{13} \leq 5）,\\
&g_{16}: \ 3.0-y_4 \geq 0 \ （即, x_{21} \leq 3）,\\
&g_{17}: \ 15.0-y_5 \geq 0 \ （即, x_{22} \leq 15）,\\
&g_{18}: \ 5.0-y_6 \geq 0 \ （即, x_{23} \leq 5）
\end{aligned}$$

EP_2 找到的最好运输费用的平均值（20 次独立运行的平均值）为 460.75，而找到的最好解（其产生的总费用值为 420.74）为

$$
\begin{array}{llll}
x_{11}=3.00 & x_{12}=2.00 & x_{13}=5.00 & x_{14}=0.00 \\
x_{21}=0.00 & x_{22}=0.00 & x_{23}=0.00 & x_{24}=15.00 \\
x_{31}=0.00 & x_{32}=18.00 & x_{33}=0.00 & x_{34}=2.00
\end{array}
$$

正如所预测的那样，EP_2 的结果比先前演化程序 EP_1 要好。EP_2 的另一优点是不必对非整数校正结果（使之移到可行区域里）。另外，EP_2 的执行结果依赖于由用户给出的搜索空间的开始点。出于这一理由，提供一个对该系统完整的分析确实十分困难。

3. 演化程序 EP_3

这里描述的第三个演化程序 EP_3 是第 7 章所述的 GENOCOP。因为 GENOCOP（这里的演化程序 EP_3）只能处理线性约束，很明显 $dom(EP_3) \subseteq dom(EP_2)$和 $EP_2 \prec EP_3$。

运输问题 P 是带有 $m=12$ 个变量的问题；每个染色体被编码成一个有十二个浮点数的向量$\langle y_1,\ldots,y_{12}\rangle$。因此问题 P 为

$$\min \sum_{i=1}^{12} f(y_i)$$

其中

$$
\begin{array}{llll}
y_1=x_{11}, & y_2=x_{12}, & y_3=x_{13}, & y_4=x_{14}, \\
y_5=x_{21}, & y_6=x_{22}, & y_7=x_{23}, & y_8=x_{24}, \\
y_9=x_{31}, & y_{10}=x_{32}, & y_{11}=x_{33}, & y_{12}=x_{34}
\end{array}
$$

六个独立的线性约束为：

$$
\begin{aligned}
& y_1+y_2+y_3+y_4=10 \\
& y_5+y_6+y_7+y_8=15 \\
& y_9+y_{10}+y_{11}+y_{12}=20 \\
& y_1+y_5+y_9=3 \\
& y_2+y_6+y_{10}=20 \\
& y_3+y_7+y_{11}=5
\end{aligned}
$$

（第七个等式 $y_4+y_8+y_{12}=10$ 是不必要的，因为它和给定的六个等式线性相关）；另一个线性不等式是

$$y_i \geq 0, \quad (i=1,\ldots,12)$$

我们运行了 20 次 GENOCOP。总的运输费用值在下面最差解（四舍五入到小数点后第二位）：

$$
\begin{array}{llll}
x_{11}=3.00 & x_{12}=4.38 & x_{13}=2.62 & x_{14}=0.00 \\
x_{21}=0.00 & x_{22}=15.00 & x_{23}=0.00 & x_{24}=0.00 \\
x_{31}=0.00 & x_{32}=0.62 & x_{33}=2.38 & x_{34}=17.00
\end{array}
$$

为 420.74，对最好解：

$$
\begin{array}{llll}
x_{11}=3.00 & x_{12}=7.00 & x_{13}=0.00 & x_{14}=0.00 \\
x_{21}=0.00 & x_{22}=13.00 & x_{23}=2.00 & x_{24}=0.00 \\
x_{31}=0.00 & x_{32}=0.00 & x_{33}=3.00 & x_{34}=17.00
\end{array}
$$

为 356.98。GENOCOP 系统返回的 20 次运行的平均运输费用为 405.45。当然，所有获得

的解都是可行的。很明显，GENOCOP 作为一个与问题更有关的系统比演化策略 EP_2 执行得好。

4. 演化程序 EP_4

下一个演化程序 EP_4 是 GENETIC-2。如第 9 章所描述的，系统的构造是用来优化任何非线性运输问题的，所以很明显也适用于 $dom(EP_4) \subseteq dom(EP_3)$ 和 $EP_3 \prec EP_4$。在 GENETIC-2 里，一个矩阵表示一个潜在解：定义了适当的算子以处理这样的表达。

我们运行了 20 次 GENETIC-2。总的运输费用值在最差解（四舍五入到小数点后第二位）：

$$
\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 5.00 & x_{13} = 0.00 & x_{14} = 2.00 \\
x_{21} = 0.00 & x_{22} = 15.00 & x_{23} = 0.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 5.00 & x_{34} = 15.00
\end{array}
$$

为 397.02 到最好解：

$$
\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 7.00 & x_{13} = 0.00 & x_{14} = 0.00 \\
x_{21} = 0.00 & x_{22} = 13.00 & x_{23} = 2.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 3.00 & x_{34} = 17.00
\end{array}
$$

为 356.98 之间变化（和 GENOCOP 找到的解相同）。但是，由 GENETIC-2 找到的 20 次运行平均运输费用为 391.65，要好于 GENOCOP 的 405.45。所有解是可行的。很明显，作为一个更与问题有关的系统，GENETIC-2（EP_4）比 GENOCOP（EP_3）要执行的好。

5. 演化程序 EP_5

最后的演化程序 EP_5 还是基于 GENESIS 1.2ucsd，是和前面所描述的实算十分相同的系统。这一次我们试图“调节”罚函数集以集中于问题 P。另外消除所有的等式，启发是：处理不等式比处理等式约束更容易。

所以，问题 P 被重写为：

$$
\begin{aligned}
\min \quad & f(y_1) + f(y_2) + f(y_3) + f(10.0 - y_1 - y_2 - y_3) + f(y_4) + f(y_5) + f(y_6) \\
& + f(15.0 - y_4 - y_5 - y_6) + f(3.0 - y_1 - y_4) + f(20.0 - y_2 - y_5) \\
& + f(5.0 - y_3 - y_6) + f(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0),
\end{aligned}
$$

其中

$$
\begin{aligned}
& y_1 = x_{11}, \quad y_2 = x_{12}, \quad y_3 = x_{13}, \\
& y_4 = x_{21}, \quad y_5 = x_{22}, \quad y_6 = x_{23}
\end{aligned}
$$

及

$$
\begin{aligned}
& 0.0 \le y_1 \le 3.0, \\
& 0.0 \le y_2 \le 10.0, \\
& 0.0 \le y_3 \le 5.0, \\
& 0.0 \le y_4 \le 3.0, \\
& 0.0 \le y_5 \le 15.0, \\
& 0.0 \le y_6 \le 5.0
\end{aligned}
$$

六个试图调节的罚函数为：

$$p_1 = \begin{cases} w_1 \cdot (c_1 + y_1 + y_2 + y_3 - 10.0)^2 & 若\ 10.0 - y_1 - y_2 - y_3 < 0.0 \\ 0.0 & 其他 \end{cases}$$

$$p_2 = \begin{cases} w_2 \cdot (c_2 + y_4 + y_5 + y_6 - 15.0)^2 & 若\ 15.0 - y_4 - y_5 - y_6 < 0.0 \\ 0.0 & 其他 \end{cases}$$

$$p_3 = \begin{cases} w_3 \cdot (c_3 + y_1 + y_4 - 3.0)^2 & 若\ 3.0 - y_1 - y_4 < 0.0 \\ 0.0 & 其他 \end{cases}$$

$$p_4 = \begin{cases} w_4 \cdot (c_4 + y_2 + y_5 - 20.0)^2 & 若\ 20.0 - y_2 - y_5 < 0.0 \\ 0.0 & 其他 \end{cases}$$

$$p_5 = \begin{cases} w_5 \cdot (c_5 + y_3 + y_6 - 5.0)^2 & 若\ 5.0 - y_3 - y_6 < 0.0 \\ 0.0 & 其他 \end{cases}$$

$$p_6 = \begin{cases} w_6 \cdot (c_6 + 80 - y_1 - y_2 - y_3 - y_4 - y_5 - y_6)^2 & 若\ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0 < 0.0 \\ 0.0 & 其他 \end{cases}$$

其中 w_i 和 c_i 为附加的权值。

通常所有的惩罚都加到目标函数上。经过许多实算后（期间分别增加和减少了相应于违反和满足约束的权值），得到下面的集合：

$c_1 = 2.5,\quad w_1 = 2.0,$

$c_2 = 0.3,\quad w_2 = 1.3,$

$c_3 = 5.0,\quad w_3 = 2.5,$

$c_4 = 5.0,\quad w_4 = 2.0,$

$c_5 = 0.2,\quad w_5 = 1.3,$

$c_6 = 0.1,\quad w_6 = 2.0$

当然，我们不认为上述权值集合表示一个最优构造，因为调节只是“手动”调节的：如果运行约束不被满足，就逐渐增加相应的权值。但是，可以观察到：

- 带有上述权值的系统 EP_5 对问题 P 确实执行得好；
- 如果通过加入另外的来源地或者目的地，或者只是变化问题特定权值 c_{ij} 的方法改变问题，演化程序 EP_5 将不产生有意义的结果。

很明显，$dom(EP_5) \subseteq dom(EP_4)$，因此 $EP_4 \prec EP_5$。

EP_5 系统的一次运行给出下面的解：

$x_{11} = 2.93\quad x_{12} = 6.91\quad x_{13} = 0.16\quad x_{14} = 0.00$

$x_{21} = 0.07\quad x_{22} = 13.09\quad x_{23} = 1.84\quad x_{24} = 0.00$

$x_{31} = 0.00\quad x_{32} = 0.00\quad x_{33} = 3.00\quad x_{34} = 17.00$

注意所有的约束均被满足，目标函数值为 391.2。上述解可以被手动校正成 GENOCOP 和 GENETIC-2 所找到的最后解：

$x_{11}=3.00$　$x_{12}=7.00$　$x_{13}=0.00$　$x_{14}=0.00$

$x_{21}=0.00$　$x_{22}=13.00$　$x_{23}=2.00$　$x_{24}=0.00$

$x_{31}=0.00$　$x_{32}=0.00$　$x_{33}=3.00$　$x_{34}=17.00$

但是，系统 EP_5 同时也找到一个更好值 391.2，即对下面运输计划的值 378.25：

$x_{11}=2.53$　$x_{12}=7.47$　$x_{13}=0.00$　$x_{14}=0.00$

$x_{21}=0.47$　$x_{22}=12.53$　$x_{23}=2.00$　$x_{24}=0.00$

$x_{31}=0.00$　$x_{32}=0.00$　$x_{33}=3.00$　$x_{34}=17.00$

它是难以校正的。记住最优解不必是由整数组成。例如，我们从 GENETIC-2 获得的一个解为：

$x_{11}=3.00$　$x_{12}=7.00$　$x_{13}=0.00$　$x_{14}=0.00$

$x_{21}=0.00$　$x_{22}=12.25$　$x_{23}=2.75$　$x_{24}=0.00$

$x_{31}=0.00$　$x_{32}=0.75$　$x_{33}=2.25$　$x_{34}=17.00$

它的总运输费用等于 380.86。

总之，有可能构造一个适合问题 P 的“完美”的演化程序。可以通过引入运输费用 C_{ij}、六个独立的约束的特征、也可是其他修正可行解的启发式方法加入其他知识到这样的系统当中。可以引入附加的约束以“指导”系统朝向应有的方向。但是应该注意构筑这样系统的难度是随问题的维数增加的，而且它的实用性将会受到限制（只适合问题 P）。

实算结果肯定了较早时候已经证实过的基于直觉的假定：与问题有关的知识在增强算法的执行结果的同时限制了其应用范围。

如前所述，出于对演化程序进行公正的比较，我们设定群体规模为 70，对所有的实算，代数为 5000，而且所有的运行都重复 20 次。但执行演化程序的初始化时，我们使用了不同的技术。第一个演化程序 EP_1 以这样的方法产生其群体：个体不必是可行的（因为约束包括等式，令人惊讶的是它甚至可以是只有一个可行个体）。第二个演化程序 EP_2 使用了一个单独（可行的）的个体作为其开始点；不同的初始可行点被产生二十次用来进行测试。第三个程序 EP_3 产生一些数（此数是该系统的一个参数）试图发现搜索空间里的一个初始可行个体。如果成功，初始群体将由找到个体的 *population_size* 个同样拷贝组成。如果不成功，系统将为用户提示一个可行初始点；这些运行初始可行点的集合和 EP_2 所使用的集合是相同的。第四个程序 EP_4 产生并维持一个可行个体的群体，而 EP_5（像 EP_1）一样产生可能是不可行个体的初始群体。

在比较我们的演化程序时，很重要的是要知道其中初始化技术是不同的；但是，实算的结果表明特定初始化技术对系统执行结果的影响是可以忽略的。这并不令人惊讶：对一个通常的高约束问题（如对特定的运输问题），搜索空间里的一个“可行点”并不意味着一个“好的”点。只有用户有一个好的能并入系统的启发式规则的情况（必须注意避免过早收敛！），一个启发式初始化才是有用的。EP_1 或 EP_5 当用可行个体来初始化时不会有较大的改进，除非一个可行个体是真正的好个体。演化程序 EP_4 中提出的“智能”初始化产生一可行点的集合，其平均评价值为 456。此初始化不增强算法：它只是简单地以可行群体开始，因为 EP_4 的算子只维持可行性。其他程序（EP_2 和 EP_3）对在区

间(493, 610)内的适应值相对较差可行点进行了校正（平均值为 562）。

结论是初始化过程不影响得到的结果。

在上述引导性的评注之后，我们准备回答两个实际的且与演化编程有关的问题：对一个给定问题 P，

（1）一个演化程序的强或弱应该是怎样的？

（2）我们应该如何构造一个演化程序？

回答这些问题是不容易的。在这里，我们将提供一些一般性的评论和基于各种实算得到的启示及一些基于期望的想法。

第一个问题是有关优化和构造演化程序的选择。对一给定问题 P，一个演化程序的弱或强应该如何？换句话说，对一个给定问题 P，我们应该构造 EP_2 还是 EP_4 呢？我们的推论建议引入与问题有关的知识对精度而言能给出更好的结果。但是如引言部分所表明的，如果进行深入的问题分析以设计特定的表达、算子，一个强的、高执行系统的开发可能耗费很长的时间。另外，我们已经有一些标准的软件包，像 Grefenstette 的 GENESIS、Whitley 的 GENITOR、Davis 的 OOGA、Schraudolph 的 GENESIS 1.2ucsd，或者是 Schwefel 的演化策略系统等。而且，如果我们试图找到一个对一给定问题的有效的二进制表达，可能只有很少的或者根本没有软件可用。

当然回答有时是肯定的，有时是否定的。如果正在解决一个带非常规强约束的运输问题（即约束必须被满足），使用一些标准的软件包获得可行解的机会就会很小。或者，如果我们以可行解的一个群体开始，并强制系统维持它们，我们可能不会得到任何进展。在这种情况下，系统不比常规搜索方法执行得更好。而对其他一些问题，这样的标准包可能得到十分满意的结果。简而言之，对此问题作决定的责任依赖于用户；决定与许多因素有关，包括解的精度要求、找到解的可行性（即问题约束的重要程度）、使用开发系统的频率、等等。

假定，出于某些理由，我们不得不（或者想要）构造一个新系统来解决一非常规的优化问题。这可能是因为标准的遗传算法包不提供可以接受的可行解且没有可用的适合此问题的计算包。我们不得不作出选择：尝试构造一个演化程序，还是采用一些传统的（经验性）的方法。通常一个传统的方法采用三个步骤解决一个优化问题：

(1) 理解问题；

(2) 解决问题；

(3) 实现先前步骤中发现的算法。

在传统的方法里，一个程序员应该指出此问题，只是产生一个正确的程序。但是常常碰到一个问题的算法解是不可能的，或者至少是非常难的情况。到了最后，对某些应用找到最优解不是最重要的，找到任何带有合理误差界限（和最优值的相对距离）的解都可以。例如，在某些运输问题里，可以只寻找一个好的运输方案，而不要求发现最优解。在我们的实算中，一些开发的演化系统建议（相当快地得到）一个解，比方说为 1109，而最优值是 1102。在这种情况下误差小于 1%，近似解可能就是更希望得到的。

一个演化编程的方法通常不包括第二步、也是最困难的一步。只有理解了此问题，我们才能够谈得上实现。在构造一个演化程序中，程序员的主要任务是选择适当的数据

编码和操作于它们的“遗传”算子。其余的留给演化过程去做。此任务不一定是琐碎的，因为和数据结构不同，它可以使用染色体表达，每种数据结构可能为遗传算子提供更宽的选择性。这可能包括程序员对问题性质的理解；但是，不必首先解决此问题。为构造一个演化程序，一个程序员应该遵循下面的五个基本步骤：

(1) 首先选择一种对此问题解的遗传表达。这要求对问题性质有一定的理解。但不是必须解决此问题。被选择的问题解的表达应该是“自然的”，这必须由程序员决定，注意，要在当前的编程环境下选择适当的自己的数据编码结构。看来这是最重要的步骤，因为它将影响演化程序的其余组件。表达应该携带所有有关解的重要信息；不幸的是对这种选择没有现成的指导规则。而且许多演化计算范例的基本不同主要集中在此（二进制串、浮点数向量、有限状态机、计算机程序等）。

(2) 程序员的第二个任务是产生一个解的初始群体，这可以用许多方法完成：随机法、经验算法的输出等。当与问题有关的约束必须满足时，需要多加注意；通常一个可行解的群体可以被作为一个演化程序好的开始点（见下一章）。但在许多情况下，我们不知道任何有关解的可行性的信息；有时是修补一个不可行解和找到一个可行解一样困难，如强约束时间表问题。

(3) 对许多优化问题，选择一个评价解适应值的评价函数不应该产生太多的困难。但有时此任务是非常复杂的，见下一章。

(4) 设计“遗传”算子时应该特别注意 — 此设计应该是基于问题本身和其约束的。这里有必要研究算子传达信息的含义。如果我们只研究常规空间的可行部分，算子应该转换一个可行解成另一个可行解。也可以使用修补算法、罚函数或者其他方法（见下一章）来处理与问题有关的约束。对许多真实世界的问题，引入局部搜索经验到一些算子里可能是更有帮助的。

(5) 程序使用的各种参数的值可能是由程序员给出。但是，在演化编程环境下的更高级版本里，这可能是由一个过程监督者控制，如[169]中所讨论的，此监督者的主要任务是调节所有的参数。越来越多的研究是朝着演化程序参数的自适应方向进行的。

上面描述了构造演化程序的要点，其基本思想是用“自然的”数据编码结构和对问题敏感的“遗传算子”。但是构造这样一个井然的系统仍然很难。一个有经验的程序员将处理这一任务；但产生的程序有可能是很无效的。为帮助用户完成这样的任务，产生一个新的由软件（特殊的编程语言）和硬件（并行计算机）支持的编程方法可能是有价值的，这就是我们期望想法的出处。

当前，在计算机科学领域里有许多不同的程序设计方法：结构化程序设计、逻辑程序设计、面向对象的程序设计、功能程序设计。它们都不完全支持演化程序的构造。新方法的目标是产生适当的工具来学习使用并行处理器框架（这里优化被理解成一个学习过程）。

我们希望发展这个思想来设计一种编程语言 PROBIOL (PROgramming in BIOLogy) 以支持 EVA 编程环境（EVA for EVolution progrAmming）。这个方法的一个重要问题是

控制发生在“演化引擎”(evolution engine)中的演化过程的程序实现。演化引擎由一个“微处理器集团”(society of microprocessors)表示[296]：其中的一些问题在[204]中进行了简要的讨论。新编程环境的主要动机是提供基于并行框架的编程工具，这是非常重要的；正如[6]中所说的：

> “并行化毫无疑问会改变我们认为且正在使用的计算机的方法。它将使我们置身于以前我们从未梦想到的对问题的解和知识的前沿上去。框架丰富的变化将使我们获得处理新旧问题的新的更有效的方法。”

第 15 章　演化程序和启发式方法

正如我们已在前面章节里所讨论的，最知名的演化程序包括遗传算法、演化规划、演化策略和遗传规划。也有许多合并了上述方法各种特征的杂化系统，因而难以分类；所以，我们把它们都看成演化程序，或演化算法，或是演化计算技术。

本书已多次重复了一个普遍接受的观点：任何解决一个问题的演化算法必须有五个基本步骤：

- 问题的解的遗传表达；
- 一种产生解的初始群体的方法；
- 评价函数，即环境，用以评价解的“适应值”；
- 在再生过程中改变子代遗传组成的“遗传”算子；
- 参数值，如群体规模、应用各种遗传算子的概率等等。

一个共识是：对一个特定的真实世界问题，要成功地实现一个演化计算，上面列出的基本步骤还要求其他启发式规则。这些启发式规则可应用于解的遗传表达、改变它们组成的“遗传”算子、各种参数值和产生初始群体的方法。这似乎对演化算法上述的五个基本要素只有评价函数是一个例外，它通常“当然地”不要求任何启发式修正。在许多情况下，选择一个评价函数的过程确实是直接的。如经典的数值和组合优化问题。因此在过去 20 年，检查了许多有难度的函数，它们经常作为选择不同方法、各种算子、不同表达的测试集。但是，一个评价函数的选择过程本身是相当复杂的，特别是当我们处理的问题同时有可行解和不可行解时，通常要引入启发式规则。本节将给出一些这样的启发式规则，并讨论它们的优缺点。

正如在引言中所表述的，所有演化程序都有相同的结构，见引言中的图 0.1，但是它们之间也有许多不同，这些不同经常隐藏在较低的抽象水平上。它们使用不同的数据编码代表其染色体表达，因此“遗传”算子也是不同的。它们有时在其基因里引入一些其他的信息来控制搜索过程。当然，也存在其他的不同：如图 0.1 (引言)中的两行：

select $P(t)$ from $P(t-1)$
alter $P(t)$

也可以以相反的次序出现：如在演化策略里，首先改变群体，然后通过选择过程构造一个新群体。而且，即使在一种特定的技术里，譬如遗传算法，也有许多添加成分和修改成分。例如，有许多选择个体生存和再生的方法。如第 4 章所述，这些方法包括（1）比例选择，其中的选择概率正比于个体的适应值；（2）分级加权方法，其中群体中的所有个体从最好到最差进行排序，且它们的选择概率在整个演化过程保持不变[①]；（3）竞争

① 例如，不管其精确的评价如何，最好个体的选择概率总是 0.15；第二最好个体的选择概率总是 0.14 等等。唯一的要求是较好的个体有较大的概率，且概率的总和等于 1。

选择，其中一定数量的个体（通常是两个）为下一代相互竞争选择：此竞争步骤重复和群体规模相等的次数。其中的每一种还有一些重要的细节。比例选择可能要求使用比例窗或者截断法，分级加权方法里又有不同分配概率的方法，如线性、非线性分布。在竞争选择方法里，竞争的大小起着重要的作用。确定代策略同样也是重要的。例如，有可能用一群后代替换整个群体，或者可能从父代群体和子代群体两个群体中选择最好个体—— 这种选择可以用确定性方法或者非确定性方法完成。同样也可能产生少数几个后代，甚至是单独一个，用它替换最差个体，基于这种代策略的系统被称为“稳态”。当然，也可以使用“精华”模型，它将从一代中为下一代保存最好个体①；这种模型对解决许多优化问题是十分有帮助的。最近，Ronald [333]通过允许适应个体选择它们配偶的方法控制选择过程，并对它们进行了实算，即所谓的选择–诱导方法。在这种方法里，个体仍基于它们的适应值被选择，但是，在生育时期，个体被允许根据自身的偏好选择配偶，这些偏好是以显式特征表示的，也可以是由显式的一部分组成。

对一特定的染色体表达，有许多种不同的遗传算子。在本文中，我们将考虑各种不同类型的变异；其中一些的变异概率依赖于代数和（或）位的位置。如第 10 章和第 11 章所描述的，许多“变异”算子引入了一些启发式局部搜索算法以增强演化算法的执行效果。除了 1 点杂交、还有 2 点、3 点杂交等，它们交换亲体染色体之间的适当数量的片断，此外，还有“均匀杂交”，它交换两个亲体的单个基因。当一个染色体是整数 $1,\ldots,n$ 的一个预变异时，也有许多方法来变异这样的染色体和杂交两个这样的染色体，如 PMX，OX，CX，ER，EER 杂交②。最近，Bui 和 Moon[56]在形式上归纳了对 n 维二进制编码的线性串杂交。

不同种类的结构、算子、选择方法清楚地表明：演化算法的一些版本对特定的问题比其他版本执行得要好；许多不同种类之间的比较在文献中有报告，如演化策略对遗传算法、1 点杂交对 2 点杂交对均匀杂交等等。结果是，在构造一个对特定问题（或一类问题）的成功演化算法时，用户需要使用一套启发式规则，它是对过去 20 年中的各种各样的系统和各种各样的问题的无数次实算总结。下一节将简要地讨论一些选择适当演化算法要素的启发式规则，而第 15.3 节将对用于评价群体中个体的启发式规则进行详细的讨论。

15.1 技术和启发式规则概述

用于一特定问题的数据编码和“遗传”算子集构成了任何演化算法的主要要素。例如，原始的遗传算法针对适应过程建模，主要操作于二进制串并使用以变异为基本算子的重组算子。变异翻转染色体中的位，杂交交换两个亲体之间的遗传材质：如果亲体是用一个五位串表达的，设为 (0,0,0,0,0) 和 (1,1,1,1,1)，在第二个元素后交叉向量将产生后代 (0,0,1,1,1) 和 (1,1,0,0,0)。

---

① 意思是，如果当前代中的最好个体由于选择或者遗传算子而丢失，系统强迫它加入下一代。

② 在多数情况下杂交只涉及两个亲体，但并不总是这样，见第 4 章。

演化策略是针对参数优化问题的方法；因此，染色体用一对浮点值向量表达个体，即 $\boldsymbol{v}=(\boldsymbol{x}, \boldsymbol{\sigma})$。其中的第一个向量 $\boldsymbol{x}$ 代表搜索空间里的一个点；第二个向量 $\boldsymbol{\sigma}$ 是标准偏差向量：变异通过用 $(\boldsymbol{x}', \boldsymbol{\sigma}')$ 替换 $\boldsymbol{v}$ 得以实现，这里

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0, \Delta\sigma)}$$

$$\boldsymbol{x}' = \boldsymbol{x} + N(0, \boldsymbol{\sigma}')$$

其中 $N(0,\boldsymbol{\sigma})$ 为一个独立的带零平均和标准偏差 $\boldsymbol{\sigma}$ 的随机 Gauss 向量，$\Delta\sigma$ 为该方法的一个参数。

最初的演化规划技术主要是针对人工智能的进化，而且以有限状态机作为个体的染色体表达。后代（新的 FSM）由亲体群体的随机变异产生。有五种可能的变异算子：输出符号的变化、状态转换的变化、状态的加入、删除以及初始态的改变（其他约束是状态的最小和最大数目）。

遗传规划技术提供了对计算机程序空间进行最好的（最适的）程序的演化搜索。

许多研究者通过“引入”与问题有关的知识来进一步改进演化算法。几篇文献讨论了初始化技术、不同的表达、解码技术（从遗传表达映射到“显式”表达）和对遗传算子使用启发式规则。这样的混合或非标准系统在演化计算界较为流行。这些用与问题有关的知识控制的系统常常比其他经典方法和标准技术执行得要好，正如前面章节所讨论的。

很少有启发式规则来指导用户对一特定的问题选择适当的数据编码和算子。人们的共识是对数值优化问题应该使用演化策略① 或者带浮点表达的遗传算法，而遗传算法的其他一些版本适合处理组合优化问题。遗传程序对分析计算机程序的规则有很大用处，演化规划技术可以成功地用在对系统行为的建模上，如囚犯困境问题，见[120]。其他一些用在真实世界问题上的演化算法常用的启发式规则是基于用与问题有关的知识来改进算法；这些与问题有关的知识被引入到染色体数据编码和特殊处理的遗传算子里。例如针对非线性映射问题构造的系统 GENETIC–2（第 9 章）使用了矩阵表达作为其染色体，并使用了与问题有关的变异（主要算子，使用概率为 0.4）和算术杂交（背景算子，使用概率为 0.05）。对此系统分类是很难的（第 9 章）：它不是真正的遗传算法，因为它只能在结果质量没有任何较大减损的情况下运行变异算子。而且，所有的矩阵元素都是浮点数。它也不是一个演化策略，因为它不能对其染色体编码结构中的任何控制参数进行编码。很明显，它也不是遗传规划或者其他演化方法。它只是针对特殊问题的一种演化技术。

另一种方法是上述方法的杂化。该技术[78]把现有的一些算法合并以提高演化系统的结果。这可以通过用其他算法的输出作为进化系统的初始群体的种子，或者通过把一些局部搜索算子并入到“遗传”算子里，或通过“借用”一些编码策略的方法实现。

这些思想中有一些是在演化过程的早期被嵌入，称之为**分散搜索**（scatter search，第 8 章）。该过程通过筛选启发式规则产生的好的解来产生初始群体。成为亲体的点与依赖于环境的权值以线性组合的方式被相互结合，这样的组合可以同时应用于多个亲体。

---

① 演化规划技术也可以归结位处理数值优化问题，见[117]。

线性组合算子通过适应性圆整过程进一步得到改进，以处理要求携带不连续值的元素。同严格的二进制元素相对比， 向量能同时操作于实型和整型。 最后，选择较好的结果并再次使用启发式规则， 此过程重复进行。 这种方法对混合的整数和组合优化是有用的。

有一些启发式规则可以用来产生初始群体：可以从一随机产生的群体开始，或者使用一些确定性算法获得的结果来初始化它，更多的可能性是在这两个极端之间。也有一些一般性的启发式规则来确定各种参数值：对许多遗传算法的应用，群体规模保持在 50 到 100 之间，杂交概率在 0.65 和 1.00 之间，变异概率在 0.001 和 0.01 之间。另外一些启发式规则经常被用在群体规模和算子操作概率随演化过程变化的情况下。

看起来没有一种演化技术足够完美（或者足够强有力）能超越问题的多样性：只有基于演化计算（即演化算法）概念的整个一族算法有这种鲁棒性。但是，成功的应用仍然是启发式方法，它和演化技术巧妙地掺混在一起。

15.2 可行解和不可行解

在演化计算方法里，评价函数只是用于连接问题和算法的。评价函数对群体中的个体进行打分：较好的个体有更多的机会生存和再生。因此有必要以一种“完美的方式”定义赋予问题以特征的评价函数。特别是，处理可行个体和不可行个体的问题应该十分小心：一个群体常常包含不可行个体，但是我们只是搜索一个可行的最优解。对可行和不可行个体，找到一个合适的评价方法经常是非常重要的：它直接影响算法的成败。

不可行个体的处理问题对用演化技术解决约束优化问题是很重要的。例如在连续域里，一般的非线性规划问题① 是找到 x 以便：

$$\text{优化} f(x),\quad x=(x_1,\ldots,x_n)\in R^n$$

其中 $x\in F\subseteq S$。集合 $S\subseteq R^n$ 定义了搜索空间，而集合 $F\subseteq S$ 定义了一个可行搜索空间。通常搜索空间 S 被定义成一个 R^n 里的 n 维矩形（变量的域由它们上下边界定义）：

$$l(i)\le x_i\le u(i),\quad 1\le i\le n$$

而可行集合 F 由 S 和另外的 $m\ge 0$ 个约束集合的交集定义：

$$g_j(x)\le 0,\ \text{其中}\ j=1,\ldots,q\ \text{和}\ h_j(x)=0,\ \text{其中}\ j=q+1,\ldots,m$$

演化计算技术对非线性编程问题的应用多数都集中于复杂的目标函数连同 $F=S$。在过去 20 年里，各个研究者使用的几个测试函数值考虑了 n 个变量的域；对 De Jong[82]建议的 F1~F5 测试函数也是这样，对此后建议的许多其他测试例子也是这样。

不连续域里的约束问题是较早就被公认的了。背包问题、集合覆盖问题、任何类型的日程表和时间表问题都是受约束的。有几种启发式方法用于处理约束；但是，这些方法没有进行系统的研究。

一般来说，搜索空间 S 由彼此分开的可行子空间 F 和不可行子空间 U 组成，见图 15.1。对这些子空间，我们没有做任何假定。特别是它们不必是凸集，且它们不必是相连

① 我们这里考虑的只是连续变量。

接的，如图 15.1 中的例子，其中搜索空间的可行部分 F 由四个分立的子集构成。在解决优化问题时，我们搜索一个可行的最优解。在搜索过程中我们不得不处理各种可行和不可行个体，如图 15.2，在演化过程中一个群体可能包含可行的个体(b, c, d, e, i , j , k, p) 和不可行的个体(a, f , g, h, l, m , n, o)，而全局最优解由“X”标记。

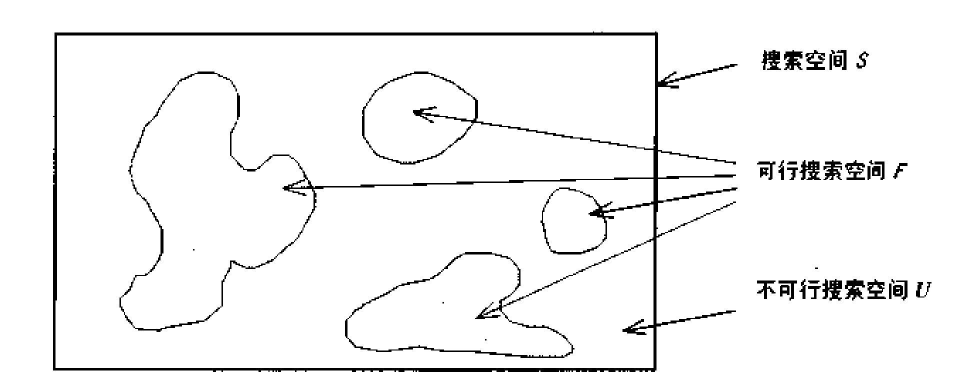


图 15.1 一个搜索空间及其可行部分和不可行部分

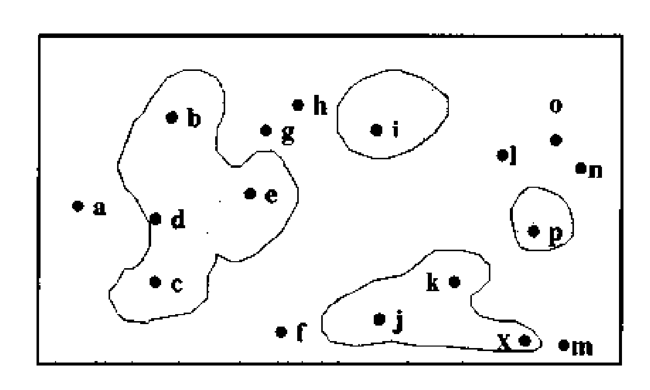


图 15.2 有 16 个个体的一个群体，a – o

群体中可行和不可行个体的存在影响着演化算法的其他部分。例如，是否精华选择方法应该考虑保存最好可行个体，或者只是总体上的最好个体呢？再者，某些算子可能只应该用于可行个体。但是这样一个方案的主要方面是必须对可行和不可行个体进行评价的，如何评价群体中的个体的问题不是一个小问题。一般情况下，我们不得不设计两个评价函数 $eval_f$ 和 $eval_u$，分别针对可行和不可行域。在这方面有许多重要的问题需要解决，我们将在下一部分对它们进行详细的讨论。这些问题有：

A. 两个可行个体应该如何被比较？如，图 15.2 的“c”和“j”。换句话说，应该如何设计函数 $eval_f$?

B. 两个不可行个体应该如何比较？如图 15.2 中的“a ”和“n ”，换句话说，应该如何设计函数 $eval_u$?

C. 函数 $eval_f$ 和 $eval_u$ 如何相联系？如对任何 $s \in F$ 及 $r \in U$，是否应该假定 $eval_f(s) \succ eval_u(r)$（符号 $\succ$ 的意思是“好于”，即对求最大问题是“大于”；对求最小问题为“小于”）？

D. 是否应该认为不可行个体是有害的，且应该从群体中消除它们？

E. 是否应该把不可行解移到可行空间中的最近点来“修补”它们？这时“m ”的修

补版本可能是最优点“X” ，见图 15.2。

F. 如果修补不可行个体，是否应该在群体中用其修补版本替换它？或者是否只是出于评价的目的使用修补步骤？

G. 因为我们的目标是找到一个可行的最优解，是否应该对不可行个体进行惩罚？

H. 是否应该从可行个体的初始群体开始，并用特殊的算子来维持后代的可行性？

I. 是否应该使用解码器改变搜索空间的拓扑结构？

J. 是否应该抽取定义可行搜索空间的约束集合并分开地处理个体和约束？

K. 是否应该集中搜索空间中的可行部分和不可行部分的边界？

L. 如何发现一个可行解？

在演化计算领域中现在有几种处理不可行解的倾向。我们在第 7 章中对数值优化环境下进行了一些讨论；这里我们讨论不连续和连续域的例子。

15.3 评价个体的启发式方法

本节将讨论几种处理群体中可行和不可行解的方法；其中的大多数都是最近推出的新方法。仅仅在几年前，Richardson 等[332]认为：“应用遗传算法到约束优化问题的尝试应该遵从下面两个不同的原则：（1）对遗传算子的修正（2）惩罚不满足所有约束的串”。许多建议的启发式规则在这已不再适用。即使用罚函数，也是几种方法组成，这些方法在罚函数设计和应用于不可行解的许多细节上是不相同的。另外一些方法有维持群体中个体的可行性的或是使用特殊的算子或解码器、或强加一个任何可行解比任何不可行解要“好”的限制，或以一个特定的线性次序来考察约束，修补不可行解，使用多目标优化技术，基于文化算法或者用一个特殊的共同精华模型来评价解等等。我们将通过回答前面部分 A～L 的问题依次讨论它们。

A. 函数 $eval_f$ 的设计

这通常是最容易的问题：多数优化问题里，对可行解的评价函数是给定的。对数值优化问题及多数运筹学问题（背包问题、货郎担问题、集合覆盖问题等等）也是这样。但是对某些问题，评价函数的选择仍然是很粗糙的。例如在构造一个演化程序来控制移动式机器人时（第 11 章），就有必要评价机器人的路径。是路径#1 还是路径 #2 （图 15.3）有

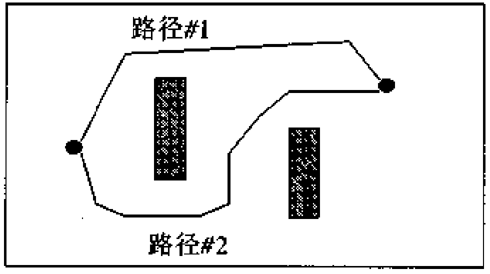


图 15.3 在一个环境下的路径

更好的评价值还不清楚，因为要考虑它们总的距离长短、障碍的大小、清洁性和光滑性

等因素：路径#1更短，但是路径#2更光滑。对这类问题，有必要将一些启发式方法引入到评价函数里。注意即使是度量一个路径光滑性和清洁性的子任务也不简单。

对许多设计问题也存在这样的情况，其中没有比较两个可行设计的明确的公式。很明显，一些与问题有关的启发式方法在这种情况下是必要的，它将提供对一个可行个体 x 的数值度量 $eval_f(x)$。

说明评价可行个体必要性的一个最好的例子是可满足性(SATisfiability, SAT)。对一给定的合取范式，设为：

$$F(x) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee x_3)$$

很难比较两个可行个体：$p = (0, 0, 0)$ 和 $q = (1, 0, 0)$ （在两种情况下 $F(p) = F(q) = 0$）。De Jong 和 Spears[90]检查了几种可能性。例如可以定义 $eval_u$ 为被评价为真的连接数的比率；在此情况下

$$eval_f(p) = 0.666 \text{ 及 } eval_f(q) = 0.333$$

也可以[305]将布尔变量 x_i 改变成浮点数 y_i，并分配：

$$eval_f(y) = |y_1 - 1||y_2 + 1||y_3 - 1| + |y_1 + 1||y_3 + 1| + |y_2 - 1||y_3 - 1|$$

或者　$$eval_f'(y) = (y_1 - 1)^2(y_2 + 1)^2(y_3 - 1)^2 + (y_1 + 1)^2(y_3 + 1)^2 + (y_2 - 1)^2(y_3 - 1)^2$$

在上述情况下，SAT 问题的解对应于目标函数的全局最小点：$F(x)$的 true 值等同于 $eval_u(y)$ 的全局最小值 0。

让我们引用[109]中的话，其中作者否定构造一个直接的 $eval_f$ 函数来解决装箱问题（BPP）的思想：

"让我们定义一个适当的对 BPP 的费用函数。目标是找到需要的最小箱数，出现在脑海中的第一个费用函数是简单地用箱数来"包裹"所有的对象。这从严格的数学观点来看是正确的，但是它通常是不实用的。确实，这样的费用函数导致搜索空间非常不友好：空间中非常少的优化点丢失在指数于这些点的数目的点中，其中主要费用函数仅仅比最优点高出一个单位。更糟的是，这些轻度的次优点产生相同的费用。麻烦是这样的费用函数缺乏指导算法搜索的任何能力，使问题成为'大海捞针'。

这样，我们就集中于下面的对 BPP 的费用函数：求最大，

$$f_{BPP} = \frac{\sum_{i=1}^{N}(F_i / C)^k}{N}$$

对 N 个箱，解中实际使用的箱数可以评价，F_i 为箱 i 中对象的总尺寸（填满的），C 为箱的容量，k 为常数，且 $k > 1$。

常数 k 表示我们集中于"最极端的"箱，而不是未填满的箱。k 越大，我们宁愿要的填满的"精华"箱越多（相对于大约相同填充度的箱集）。实际上，k 的值使我们可以改变优化函数的"粗糙度"，从"大海捞针"（$k = 1$，$f_{BPP} = 1/N$）达到"最佳填充箱"（$k \to \infty$，$f_{BPP} \to \max_i [(F_i/C)^k]$）。"

很明显，选择"完美的" $eval_f$ 的过程绝不是不重要的。

还有另一种可能性：在某种情况下我们根本不必定义评价函数 $eval_f$! 此函数只有当演化算法使用比率选择时才是必需的（见第 4 章）。对其他类型的选择路线，有可能建立的群体中个体只是线性排序关系。如果一个线性次序关系ρ处理“是否可行个体 x 要比可行个体 y 好？”一类的决定[①]，那么仅有这样的关系的ρ对竞争和加权选择方法是充分的，这些方法要么要求从一些数量的个体中选择最好个体，要么要求所有个体的线性分级。

当然，也可能需要使用一些启发式规则来建立这样的线性次序关系ρ。例如，对多目标优化问题，建立一个个体解之间的偏序是相当容易的；另外一些启发式规则可能对用偏关系无法比较的个体的排序是必要的。

总之，似乎竞争和分级加权选择对用户更灵活一些：有时，比较两个解比提供评价值的数值更容易。但是，在这些方法中，有必要解决比较两个不可行个体的问题（见 B）及比较可行个体和不可行个体的问题（见 C）。

B. 函数 $eval_u$ 的设计

这确实是一个十分难的问题。我们可以通过排除不可行个体来避免它（见 D）。有时可以控制函数 $eval_f$的域以处理不可行个体，即 $eval_u(x) = eval_f(x) \pm Q(x)$，其中 $Q(x)$表示一个对不可行个体 x 的罚函数，或者对这样个体的一个修补费用（见 G）。另一个选择是设计分离的评价函数 $eval_u$，独立的 $eval_f$，但是在这种情况下我们不得不建立一些这两个函数之间的关系（见 C）。

不可行个体是很难评价的。对背包问题更是这样。其中容量违背的量并不需要是一个好的度量个体“适应值”的方法（见 G）。这对许多日程表和时间表问题，以及路径安排问题都同样存在：是路径#1 好，还是路径#2 好（图 15.4）还不十分清楚，因为路径#2 和障碍有更多的交叉点而且比路径#1 长；另一方面，使用上述准则，最不可行路径比直线路径“差”（路径#1）。

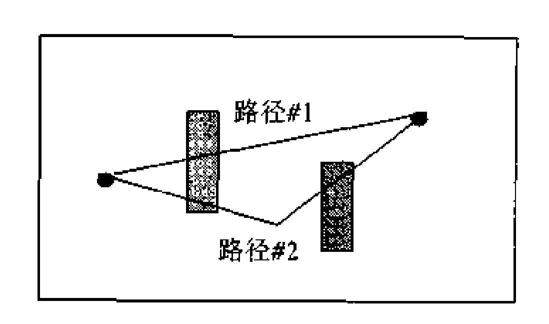


图 15.4 一个环境里的不可行路径

正如可行解的情况（A），有可能发展一种对不可行个体的排序关系（相对于构造 $eval_u$）；在两种情况下，有必要建立一个可行个体和不可行个体的评价方法之间的关系(C)。

① 表达 $\rho(x, y)$可以被解释为 x 比 y 好，其中 x 和 y 是可行的。

C. 函数 $eval_f$ 和 $eval_u$ 之间的关系

假定我们在群体中既处理可行个体，也处理不可行个体。而且我们用两个评价函数 $eval_f$ 和 $eval_u$ 来评价它们。也就是对一个可行个体 x 的评价和对一个不可行个体 y 的评价分别为 $eval_f(x)$ 和 $eval_u(y)$。现在很重要的是建立这两个评价函数之间的关系。

如 B 中所描述的,一种可能是用 $eval_f$ 的方法设计 $eval_u$，即 $eval_u(y) = eval_f(y) \pm Q(y)$，其中 $Q(y)$ 或者表示对不可行个体 y 的惩罚，或者表示对这样一个个体的修补费用（我们在 G 中讨论这个选项）。

另一个可能是，我们可以构造一个全局评价函数 $eval$ 如下：

$$eval(p) = \begin{cases} q_1 \cdot eval_f(p) & 若\ p \in F \\ q_2 \cdot eval_u(p) & 若\ p \in U \end{cases}$$

换句话说，用两个权值 q_1 和 q_2 来分别对 $eval_f$ 和 $eval_u$ 的重要性打分。

上面所描述的方法都允许不可行个体比可行个体“更好”。一般而言，有可能同时有一个可行个体 x 和一个不可行个体 y，使 $eval(y) \succ eval(x)$ 成立①，这可能引起算法收敛到不可行解；这就是为什么几位研究者实算了动态惩罚 Q（见 G），以增加不可行个体相应于当前搜索状态的压力。这些方法的弱点是它们的问题相关性：选择 $Q(x)$ 或者权值 q_1 和 q_2 通常和解决原始问题一样难。

另外，一些研究者[312,279]报告了用他们的演化算法得到的较好结果，他们的算法都假设任何可行个体都比任何不可行个体好。Powell 和 Skolnick[312]应用此启发式规则于数值优化问题（见第 7 章）：可行解的评价被映射到区间$(-\infty,1)$里；不可行解到区间$(1,\infty)$（对求最大问题）。Michalewicz 和 Xiao[279]实算了路径安排问题（第 11 章）并对可行个体和不可行个体使用了两个分离的评价函数。$eval_u$ 的值通过加入一个常数而增加（即减小吸引力），以便使最好的不可行个体比最差的可行个体差。但是，不清楚的是是否总是有这种情况。特别值得怀疑的是是否可行个体“b”（图 15.2）应该比不可行个体“m”有更高的评价值，因为它“紧邻”最优解。相似的例子可以从路径安排问题中获得：不清楚的是是否一个可行路径#2 （见图 15.5）应该比不可行路径#1 享有更高的评价值。

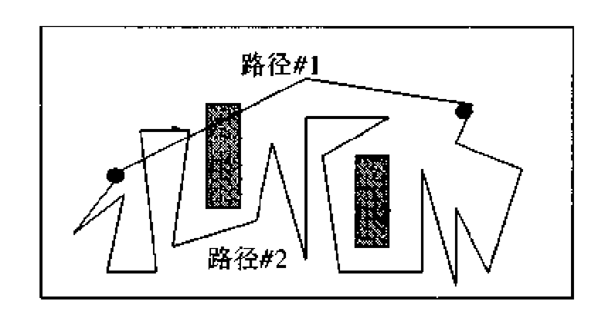


图 15.5 一个环境中的不可行和可行路径

当应用演化算法于一个特定问题时，在可行个体和不可行个体评价函数之间建立一种关系是最具挑战性的问题之一。

① 符号 $\succ$ 的意思是“好于”，即对求最大为“大于”；对求最小为“小于”。

D. 不可行个体的丢弃

“死亡惩罚”的启发式规则在许多演化技术中是常用的方法，如演化策略。注意，消除不可行个体使算法得到某种简化：例如无需设计 $eval_u$ 并将它和 $eval_f$ 进行比较。

当可行搜索空间为凸集而且它是整个搜索空间的一个合理部分时，从一个群体中消去不可行解的方法可能能工作得很好，像演化策略不允许等式约束，因为有这样的约束，可行搜索空间和不可行搜索空间大小的比率为零。否则这样的方法就有严格的限制。例如，对许多搜索问题，初始群体只由不可行个体组成，改进它们，而不是放弃它们可能是必要的。而且，如果有可能“交叉”一个不可行区域（特别是对非凸可行搜索空间）时，系统常常更容易达到最优解。

E. 不可行个体的修补

在演化计算界，修补算法的应用十分广泛：对许多组合优化问题，如货郎担问题、背包问题、集合覆盖问题等。很容易“修补”一个不可行个体。这样的修补版本也可以只用于评价，即

$$eval_u(y) = eval_f(x)$$

其中 x 为 y 的修补版本（可行的），或者可以用来以某种概率替换群体中的原始个体（见F）。注意，解“m”（图 15.2）的修补版本可能是最优点“X”。

修补不可行个体过程是和学习与评价的结合有关的，即所谓的 Baldwin 效应[399]。正如一般的局部搜索，特别是对最近可行解的局部搜索，学习和评价彼此相互作用，个体获得改进的适应值。在这种方法里，局部搜索的方法和一特定串一个代的学习十分相似。

这些方法的缺点是它们对问题的依赖性。对某个特定的问题，应该设计一种特殊的修补算法。而且没有通用的启发式规则来设计这样的算法：通常可使用贪婪修补、随机修补或者一些其他能指导修补过程的启发式规则。对一些问题，修补不可行个体的过程也可能和解决原始问题一样复杂。如对非线性约束问题、计划和时间表问题和一些其他的问题。而处理非线性约束的数值优化的 GENOCOP III 系统（第 7 章）也是基于修补算法的。

F. 用修补版本替换原来的个体

这个问题和所谓的 Lamarck 进化[399]有关，它假定一个个体在其生命期间改进，最终的改进被编码到染色体中去。正如[399]中所描述的：

> “我们的分析和经验结果表明，Lamarckian 策略经常是一个非常快的搜索形式。但是也存在这样的函数，没有学习的简单遗传算法和 Lamarckian 策略常...收敛到局部最优，而利用了 Baldwin 效应的简单的遗传算法收敛到全局最优。”

这就是为什么在使用替换策略时要非常小心。

正如第 4.5.2 节所讨论的，Orvosh 和 Davis 报告了一个所谓的 5%法则，它表明在许多组合优化问题中，使用修补算法的演化计算技术时，当用 5%的修补个体替换它们不可行的原始个体时，可能获得最好结果。在连续域里，有一种新的修补规则。对带有非线

性约束的数值优化问题，GENOCOP III 系统（第 7 章）是基于修补方法的。第一个实验（基于带有各种数量的变量、约束、约束类型、最优时的激活态约束等的 10 次测试）表明 15%替换规则的系统很明显好于替换率高于或者低于此替换率的情况。

看来“最优”的替换概率也与问题有关，并可能随演化过程而变化。进一步的研究需要比较设定此参数时的不同启发式规则，它对于所有基于修补方法都是十分重要的。

G. 惩罚不可行个体

这在遗传算法界是常用的方法。函数 $eval_f$ 的域被扩展；该方法假定

$$eval_u(p) = eval_f(p) \pm Q(p)$$

其中 $Q(p)$是一个对不可行个体 p 的惩罚项，或者是修补这样一个个体的费用。主要问题在于，惩罚函数 $Q(p)$应该如何设计？直觉上很简单：惩罚应该尽可能的低，仅有的限制应该是不可行解不能是最优，即所谓的最小惩罚规则[239]。但是有效地实现这一规则很难做到。

不可行个体“p”和搜索空间 S 的可行部分 F 在惩罚这样的个体时扮演着很重要的角色：一个个体可能只是被作为不可行个体被惩罚，度量其不可行性的“量”以确定惩罚值，或者应该考虑“修补”个体的费力程度。例如对问题大小为 99 的背包问题，我们可能有两个产生相同利益的不可行解，其中所有项的总权值分别为 100 和 105。但是很难得出结论：总权值为 100 的第一个个体比总权值为 105 的第二个个体“好”，尽管事实上此个体的违反约束的程度要小于另一个。理由是第一个解可能包含 5 项，每项权值为 20，而第二个解可能包含一个权值为 6 的低利益项 — 这些项的移去将产生一个可行解，所以它可能更好于第一个个体的任何修补版本。但是，在这种情况下罚函数应该考虑“最容易地修补”个体，以及其修补版本的质量；设计这样的罚函数是与问题有关的，而且通常是很难的。

在第 7 章，我们讨论了一些基于罚函数的方法：包括静态法[195]、动态法[210,267]和适应法[360,27]。也讨论了分隔遗传算法[239]，其中低的和高的惩罚被用于两个群体，它们的运行是“并行的”。

看来适当的惩罚方法的选择依赖于：（1）可行空间和整个搜索空间的大小之间的比率；（2）可行空间的拓扑性质；（3）目标函数的类型；（4）变量数；（5）约束数目；（6）约束类型；（7）在最优时激活的约束数目。因此，使用罚函数十分复杂，而且只能对它们的性质进行部分的分析。应用罚函数的一个很有希望的方向是使用自适应惩罚：把罚因子引入到染色体编码结构里去，这类似于在演化策略和演化规划的编码结构中使用一些控制参数。

H. 用特殊的表达和遗传算子维持可行群体

处理可行性问题的一个合理的启发式规则是使用特殊的表达和算子以维持群体中个体的可行性。这就是反映在标题中的这几个段落的原始思想。

过去十年中，开发了几个特殊的系统来处理特殊的优化问题；这些系统使用了独特的染色体表达和改变它们组成的特殊的“遗传”算子。在[78]中描述了一些这样的系统：

许多其他的例子在本文中也有描述。例如 GENOCOP 系统（第 7 章）只假定线性约束和一个可行的开始点（或者可行的初始群体）。算子的一个闭集维持解的可行性。例如当一个特定的解向量 x 的一个特定元素 x_i 被变异，系统确定其当前域 $dom(x_i)$（为线性约束和解向量 x 的剩余值的函数），而且 x_i 的新值从域中取出（对均匀变异以均匀概率分布，或非均匀和边界变异以其他概率分布）。在任何情况下，后代的解向量总是可行的。两个可行解向量 x 和 y 的算术杂交也类似

$$a x + (1-a)\overline{Y}$$

对 $0 \le a \le 1$，在凸搜索空间总产生一个可行解，系统假定只有线性约束，这意味着可行搜索空间 F 为凸空间。因此，没有必要定义函数 $eval_u$；函数 $eval_f$ 通常就是目标函数 f 。

这样的系统通常比其他基于惩罚方法的演化技术更可靠（见第 14 章），因此，这成为十分常见的倾向。许多研究者用与问题有关的表达和特殊算子在许多领域构造了非常成功的演化算法。这包括数值优化、机器学习、优化控制、认知模型、古典的操作研究问题（货郎担问题、背包问题、运输问题、分配问题、装箱问题、日程表问题、分区问题等等）、工程设计、系统集成、路线游戏、机器人、信号处理及许多其他问题。

同时值得注意的是，原始的演化规划技术[126]和遗传规划技术[231]也归属于演化算法一类：这些技术通过特殊的表达和算子保持有限状态机或等级结构化程序的可行性。

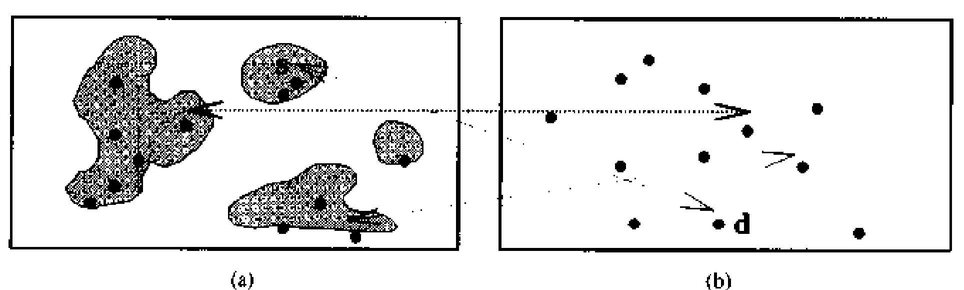
I. 使用解码器

解码器为所有研究演化技术的人们提供了一个有趣的选择。在使用解码器时，一个染色体对如何构造一个可行解“给出指令”。例如对背包问题的一个项序列可以被解释为“如果可能就取一个项”。这样的解释将总是产生可行解。考虑下面的情况：我们试图解决带 n 个项的 0–1 背包问题；第 i 项的利益和重量分别为 p_i 和 w_i。我们可以按照 p_i / w_i 降序排列所有的项，并按照下面的方法解释二进制串：

(1100110001001110101001010111010101...0010)

从表中取第一项（即取权值和利益比率最大的一项），如果该项被填入背包。接着是对第二、第五、第六、第十项等等，直到背包被填满或者没有更多的可用项为止。注意都是 1 的序列对应于一个贪婪解。如果位的序列将被翻译成一个可行解，每个可行解可能有许多可能的代码。我们可以应用经典的二进制算子（杂交和变异）：任何后代很明显都是可行的。

但是，很重要的是当使用解码器时，应该考虑几个因素。每个解码器都在一个可行解和解码解之间强加一个关系 T，见图 15.6。



(a) (b)

图 15.6 在原始（a）的和解码器（b）空间里的转换 T

有几个重要的条件必需满足：（1）对每个解 $s \in F$，有一个解码解 d；（2）每个解码解 d 对应于一个可行解 s；（3）F 中的所有解应该被相同数量的解码 d 表示。另外还有一些合理的要求是：（4）变换 T 要计算得快；（5）其局部特征是：解码的解上小的变换对解本身只产生小的变化。Palmer 和 Kershenbaum[304]报告了对遗传算法 d 的编码树有趣的研究，其中上述条件进行了公式化的表达。

J. 个体和约束的分离

这是一个一般而有趣的启发式规则。第一种规则包括多目标优化方法的利用，其中目标函数 f 和违约度 f_j（对 m 个约束）构成了一个$(m+1)$维的向量 $\boldsymbol{v}$：

$$\boldsymbol{v} = (f, f_1, \ldots, f_m)$$

使用一些多目标优化方法，我们可以尝试求其元素的最小化：对 $1 \le i \le m$，一个理想的解 x 应该有 $f(x) = 0$，及对所有可行 y，应该有 $f(x) \le f(y)$（求最小问题）。正如第 7 章所论述的，[376]是这种方法的一个成功的实现。

另一个启发式规则基于按照一特定次序处理约束的思想：Schoenauer 和 Xanthakis[346]称这种方法为“行为记忆”(behavioral memory)法。此方法初始步骤是在可行区间取样；只有在最后一步才优化目标函数 f，见第 7 章。

在第 7 章里，我们也简单地讨论了引入有关约束的知识到文化算法信念空间里的可能性[325]；这样的算法可能对在可行搜索空间里进行有效搜索提供指导[329]。

K. 利用搜索空间可行部分和不可行部分之间的边界

对约束优化问题，最新开发的一个方法是*策略震荡*(strategic oscillation)。策略震荡最初是和分散搜索演化策略一起提出的，直到最近才用于各种组合和非线性优化问题。见 Glover[147]的综述。该方法的思想是识别一个*临界水平*，它表示在可行性和不可行性之间的一个边界，但是它也可将这样的元素包含进构造阶段或者对函数值间隔的选择阶段。在可行性或不可行性环境里，基本的策略是接近并跨过可行性边界，实现的方法是通过适应惩罚和诱导（它们根据当前搜索的方向或是移到一特定区域的更深层里，或是朝着边界移动逐渐放松或者收缩），或是通过简单地使用修正梯度或者是使子梯度朝想要的方向进步。在相邻搜索环境下，选择移动规则的代价是必须考虑区域跨越和跨越的方向。从不同方向重复接近和跨越可行性边界时，可以用记忆和概率机制来避免搜索绕回到先前的轨迹。

不同规则的应用（根据区域和方向）通常伴随着跨越一边界的不同深度和不同的侧面。一种方法是接近边界并从边界撤退，并保持在单个侧面，而不跨越。单边震荡在各种计划和图论环境下很适用，其中一个有用的结构可以维持到某种程度，然后丢失掉（正如通过工作的耗尽来分配或者通过远离度定义树或旅行的条件，等等）。在这种情况下，一个建立临界水平的构造过程伴随着拆除结构的析构过程。

在策略震荡中，耗费在靠近边界区域上额外的搜索时间常常是很重要的。这可通过引入边界左右一个紧凑的震荡序列作为较大深度的大震荡前奏来解决。如果容许使用较多的力量执行每一步，该方法可能使用更多的精细移动（如各种形式的“交换”）来较

长时间地呆在边界上。例如，每当接近到达一边界时，这样的移动可以用于搜索一个局部最优点。在另外的水平上应用这样的移动的方法是*近似最佳原理*(proximate optimality principle)，该原理大致意思是：在一个水平上好的构造可能在另一水平上也是接近好的构造。

一个有用的策略震荡是增加或减少一个函数 $g(x)$的跳跃。这样的方法对许多应用是有效的，其中 $g(x)$表示劳力分配和函数值（也可以是可行性或不可行性水平），用来指导在相关区域里以各种深度进行搜寻。参考可行性和不可行性程度，$g(x)$可表示为与一套问题约束（可能表示为 $g(x)\le b$）相联系的一个向量值函数。在这种情况下，通过边界 $g(x)$控制搜索可以看成是对被选择约束集的参数化操作。经常使用的一个方法是使 $g(x)$成为一个 Lagrange 或者代理约束罚函数，而不是向量值函数，并允许根据它们的重要程度在不同元素的约束违反度之间进行交换。代理约束的方法对隔离这种交换或者是用特定的内存跟踪显示约束相对影响的行为足迹都是特别有用的。在[133], [136], [221], [393], [410], [148]可以找到嵌入这种思想的方法。

L. 发现可行解

有一类问题，其中每个可行解都是有价值的。对这样的问题，我们不必非常关心优化问题（*发现最好的可行解*），而应该试图发现在可行搜索空间 F 上的任何点。这样的问题被称为*约束满足问题*(constraint satisfaction problem)。这类问题的一个经典例子是知名的 N 个皇后问题(N-queens problem)，要求把 N 个皇后摆到一张有 N 行和 N 列的棋盘上，使任何两个皇后都不相互进攻。

几位研究者使用了各种进化系统来解决约束满足问题。Bowen 和 Dozier[48]设计了一个杂化的带有复杂染色体编码结构的遗传算法。每个染色体，除了所有变量值 v_i 以外，还存储了（1）所谓的支点（它告诉基于启发式的变异，哪个基因要变异）；（2）一个个体的“家族数”；（3）表示当分配一个值 v_i 时违反约束数目的一个向量（一个基因一个数）；（4）用来确定哪个基因应该是支点的一些启发式值。该系统成功地测试了许多随机产生的约束满足问题[48]，并比较了其他约束规划技术。

Eiben 等[99]通过设计特殊的算子处理了这一问题：作者研究了变异和杂交的几个启发式规则（包括多亲杂交）。变异算子选择亲体染色体上的许多位置并为这些位置选择新值，其中修正值的数、识别被修正值的位置的准则、定义新值的准则都是此算子的参数。多亲体杂交的主要方法是扫描，它将连续地检查亲体并为子代在标记的位置上选择一个值。带有这样的启发式算子的系统已获得实现并测试了 N 个皇后问题和图着色问题。

Paredis 实算了对约束满足问题的两种不同的方法。第一个方法[306, 307]是基于个体的更聪明的表达：在域值的顶层，每个基因允许被取出附加值“?”，它表示左开(left open)的选择。初始的群体由全为“?”的串组成：一个选择–分配–繁殖的迭代用适当域里的值替换一些“?”（通过选择一个其所在域有不止一个可用元素的变量来实现），从域里选择一个值并分配给此变量，最后执行繁殖步骤，它确保了域和当前的分配一致。这样，部分定义的个体适应值被定义成当从一个给定的部分个体开始搜索时发现的最完整解的目标函数值。该系统已经实现，并实算了几个 N 个皇后问题[307]和一些日程表问题[306]。

在第二个方法里[308]，Paredis 研究了一个共同演化模型，其中一个潜在解的群体和一约束群体共同进化：较适的解满足更多的约束，而较适的约束被更多的解违反。意思是，群体中的个体在整个搜索空间被考虑，在可行个体和不可行个体之间不加区分。一个个体的评价是在违约度 f_j 基础上确定的；但是，较好的 f_j （即激活态约束）有助于解的评价。此方法测试了 N 个皇后问题，并和其他单群体方法进行了比较[309,308]。

第16章　结　　论

演化计算领域最近几年的发展十分迅速，但仍然有许多缺口需要填补、许多实算需要去做、许多问题需要回答。在最后这一章里，我们将讨论几个重要的方向，其中我们可以预计有许多活动和重要的结果。

1．理论基础

正如本书所描述的，某些演化程序有一些理论基础：对应用于规则问题的演化策略（第 8 章），早就有收敛性质。此外，遗传算法的模式定理（第 3 章）可以解释它们的工作原理。但是当应用于特定的真实世界问题时，许多技术必须被修正。例如，为在函数优化里使用遗传算法，有必要用其他的性质扩展它们，如动态比率适应、等级比例选择、包括精华策略、搜索中采用的各种参数、各种表达、新的算子等等。演化策略当应用于约束的数值优化问题时，通常引入一些启发式方法来处理约束。虽然多数这些修正都使一个简单的算法远离其理论根基，但大多数通常都明显地增强了系统的性能。在遗传算法环境下，这些修正[89]是：

“……已经将简单的遗传算法的应用推向离其最初的理论和知识越来越远的地步，这就产生了重新认识和扩展它们的必要。”

正如本书第三版序言所指出的，对演化计算领域的研究者来说，这可能是最具挑战性的任务之一。

继续进行有关影响演化系统解决各种问题（通常是优化问题）的能力研究也是非常重要的。是哪些因素使得一个问题难以或者易于用演化方法来解决呢？这是演化计算的一个基本问题：一些有关欺骗问题、粗燥适应图、上位问题、皇家大道函数问题的结果正趋向于近似回答这一挑战性的问题。

2．函数优化

许多年来，在函数优化域里对多数演化技术进行了评价和比较。看来函数优化的这一领域始终是许多比较各种新算法新特征的测试床。可以预测，将会出现对函数优化新的演化技术的理论，如，饲者遗传算法[289]。另外，我们将看到在下面几方面的进展：

- 约束处理技术的开发。这总的来说是一个非常重要的领域，特别是对函数优化。因为多数真实的函数优化问题都包含约束。但是到目前为止，只有相当少的技术被建议、分析及相互比较。
- 用于大规模问题系统的开发。直到现在，多数实算假定变量数相当少。分析演化技术如何随有上千个变量的问题而变化的工作是有趣的。
- 处理数学规划问题系统的开发。在这一领域只做了很少的工作。有必要研究演化

系统以处理整数、布尔变量，实算混合规划和整数规划问题。

3．表达和算子

传统地，遗传算法以二进制串工作，演化策略(ES)是用浮点向量，演化规划(EP)用表达成矩阵的有限状态机，而遗传规划(GP)技术使用树作为个体的编码结构。但是，有必要系统化地研究下面的命题：

- 变大小的复数、非线性目标表达式，特别是，复杂对象的“蓝图”表达式；
- 对这些对象在因子水平的演化算子的开发。

这个研究方向可以被看成一个迈向构造并引入其他搜索技术的复杂的杂化系统的一步。例如，Lamarck 算子就很值得进行实算研究，它将在其生命期间改进一个个体。因此该个体被改进的“学习”特征将被传递到下一代。

另一些要解决的问题与对各种因素影响的理解有关（表达和算子是两个主要因素），它将影响演化方法的性能。特别是我们接近于得到下面一些问题的答案：

- 哪一类问题用演化技术是容易解决的？
- 哪一类问题用演化技术是难以解决的？
- 为什么？
- 如何对一特定问题选择特定的最好表达和算子？

对欺骗[52]、皇家大道函数[192,212]、算子的性质[315～317]、或者适应距离校正[214]等问题的研究正是朝着这一方向进行的。

4．非随机交配

当前多数引入杂交算子的技术都使用随机交配，即个体是随机配对和交配的。随着从简单到复杂系统的进步趋向，非随机配对的问题显得越来越重要，并有许多开发的可能途径：包括引入个体之间性别或者“家族”关系，或者建立一些偏向，即引诱[333]。几位研究者已经研究了一些简单的方案，如 Eshelman 的近亲防止技术[105]，但最终目标看来应该是非随机交配的演化规则。在多峰（模式）优化的环境里已开发了几种可能途径（见 8.3.1 小节），这包括共享函数，它允许构造一个稳定的子群体，及标志(tagging)，其中个体被赋予标记。但是对复杂染色体编码的研究很少有朝这一方向进行的。

5．自适应系统

因为演化算法实现了进化的思想，预计这些技术的某些自适应特征是再自然不过的事了。和引入一些控制参数到解向量里的演化策略不同，多数其他技术使用固定的表达、算子和控制参数。基于自适应机制的系统中的下述课题是最有希望的研究领域：

- 个体表达　例如 Shaefer[354]所建议的。动态参数编码技术[347]和散乱遗传算法[155](messy genetic algorithms)也归属此类。
- 算子　很明显，不同的算子在演化过程的不同阶段起着不同的作用。算子应该是适应的，如适应杂交[341,367]。对随时间变化的适应图更是如此。

- 控制参数　已经有针对此问题的实算：适应群体规模[12]或者算子的适应概率[341,367]。但是，仍然有许多事要做。

这是一个最有希望的研究方向。总之，演化算法的威力正是出于它们的适应性。

6．协同演化系统

人们对协同演化系统越来越有兴趣，其中发生不止一个演化过程：通常存在着相互作用的不同的群体，例如另外的寄生群体和掠夺群体。在这样的系统中，对某个群体的评价函数可能依赖于其他群体演化过程所处的状态。这对模拟人工生命和一些商业应用等是十分重要的话题。

协同演化系统可能对解决大规模问题是很有用的[311]，其中一个大的问题被分解成小的子问题；对每个子问题有一个单独的演化过程，同时这些演化过程是相互连接的。通常群体中个体的演化也依赖于其他群体。

在第 7 章中已经出现过一个协同演化算法（GENOCOP III），其中两个群体（并不必须是可行搜索点和完全可行参考点）相互共存。在该系统中，搜索点的评价依赖于参考点的当前群体。在同一章里，我们给出了由 Le Riche 等[239]建议的方法，其中个体的两个群体相互合作且从两个方向产生一个可行、优化解（从搜索空间的可行部分和不可行部分）。此外，Paredis[307]在约束满足问题的环境下实算了协同演化系统，见 7.4 节和 15.3 节的 L。

最近，一个协同演化系统[295]用来模拟两个竞争公司（公共汽车和有轨电车公司）在同一条路线上争夺旅客的策略。很明显，一个公司的利益依赖于另一个公司当前的策略（运输量和价格）；还研究了各种策略随时间的相互关系。

7．双倍体/多倍体对单倍体结构

双倍体或多倍体可以被看成一种把协同记忆引入到个体编码结构中去的方法。不是用单个染色体（单倍体结构）表达一个个体的精确信息。一个双倍体编码结构由一对染色体组成：两个值之间的选择由一些优势函数决定。双倍体（多倍体）编码结构在非静止环境（即随时间变化的目标函数）里与模拟复杂系统时有着特殊的重要性，这时就有可能使用协同演化模型。但是对将优势函数并入到系统中还没有理论支持，在此领域里，目前确实只做了很少的实算。

8．并行模型

将并行化用到解问题上，这在以前确实是难以捉摸的。很明显，这是计算机科学的一个重要领域。正如 Goldberg[154]所观察到的，演化算法非常适合于并行实现：

“在序列算法领域中通常并行地穿过无数的陷井和扭曲，遗传算法（高度并行算法）也同样序贯地通过不自然的陷井和弯路真是一个不小的讽刺。”

但是，把并行的思想引入到遗传算法里还没有标准的方法，已有的并行实现可分成下面几类：

- 大规模并行遗传算法　这种算法使用了大量的处理器，通常是 2^{10} 或者更多。一个单个的处理器经常被分配到群体中的一个个体上。在这种模型里，有许多可能的选择方法和配对方法（杂交重组串）。Mühlenbein[286]报告了一些在此领域里的实算工作。
- 并行岛模型(parallel island models)　这种算法假定几个子群体并行地演化。模型中包括移民的概念（一个个体串从一个子群体移到另一个），以及不同子群体中个体之间的杂交的概念。对并行岛模型有许多实算报告，有关详细的讨论，读者可以参考 Whitley 的工作[396]。
- 并行杂化遗传算法　它和大规模型并行遗传算法相似，其中处理器和个体也是一一对应的。但是只使用少数处理器。另外，这种算法合并了其他启发式算法（如爬山法）来改进系统的性能。文献中有许多这类实算的结果[286,164]，但是对这种系统的分析还很粗糙。

可以自然地把演化计算的其他范例镶入到并行模型中去，像非随机配对、一些自适应或协同演化系统。

9．其他发展

在演化计算领域，还有许多其他有趣的发展，其中之一就是所谓的文化算法(Cultural Algorithms)，可见 7.4 节。正如[330]中所描述的：

"文化算法支持两个继承的模型，一个是继承在微进化水平的特征，另一个是继承在微进化水平上的信念继承。两个模型通过通讯渠道相互作用，以确保个体的行为来改变信念结构，同时允许信念结构来约束个体的行为方式。这样，文化算法可以被描述为由三个基本元素组成：信念结构(belief structure)、群体结构和通讯渠道(communication channel)。"

文化算法的基本结构如图 16.1 所示。

```
procedure cultural algorithm
begin
    t ← 0
    initialize population P(t)
    initialize belief network B(t)
    initialize communication channel
    evaluate P(t)
    while ( not termination-condition) do
    begin
        communicate (P(t), B(t))
        adjust B(t)
        communicate (B(t), P(t))
        modulate fitness P(t)
        t ← t+1
        select P(t) from P(t-1)
        evolve P(t)
        evaluate P(t)
    end
end
```

图 16.1　文化算法的结构

此外，文化算法看来适合于控制“进化之进化”的过程。“进化之进化”的意思是进化空间中的一些元素（正如在社会进化中的一个人，或者生物进化中的一个基因，或者事物进化中的一些单元）通过它们在进化过程中的经历来改进它们的学习能力。对于把适应进化过程抽象化来说，这种思想是对生物物种遗传进化和人类社会的文化进化之间相类似思想的一般化。这种类似由 Richard Dawkins 在其书[81]中进行了精彩的探究。可以在图 16.2 中观察到支持上述思想的一种观点。第一列象征宇宙从开始到现在的时间。其他的每个列是在前面一列最后 20%的放大。星点表示进化过程中导致现代文明的主要事件。要特别注意，由星点给出的曲线增长大大快于指数函数，设为 5^x，因为如果星点在同一水平，增长将精确地为 5^x。

让我们引用[371]中的话：

“多数物理学家、化学家和生物进化论者认为有机分子的进化开始于 40 亿年前。第一个有生命的细胞出现于 35 亿年前，第一个简单的多细胞动物大约出现于 6 亿年前。猿和人类共同的祖先出现于大约 6 百万年前，而有记录的历史大约开始于 6000 年前”。

很明显，这一领域值得进行更进一步的研究。在[60, 326, 327, 328, 330, 331, 377]中描述了各种应用文化算法于各种问题的有趣的结果。

值得注意的是有许多其他学习、优化和解决问题的方法，都是基于其他来自于自然的启示，最知名的例子包括神经网络和模拟退火。人们对所有这些领域都有越来越浓厚的兴趣。许多成果丰硕同时又是极具挑战性的研究方向看来像是当前散落在各个领域的一些思想的“重组”。正如 Schwefel 和 M™nner 在第一界来自自然的并行解决问题论坛会(the proceedings of the First Workshop on Parallel Problem Solving from Nature)的引言中所写的[351]：

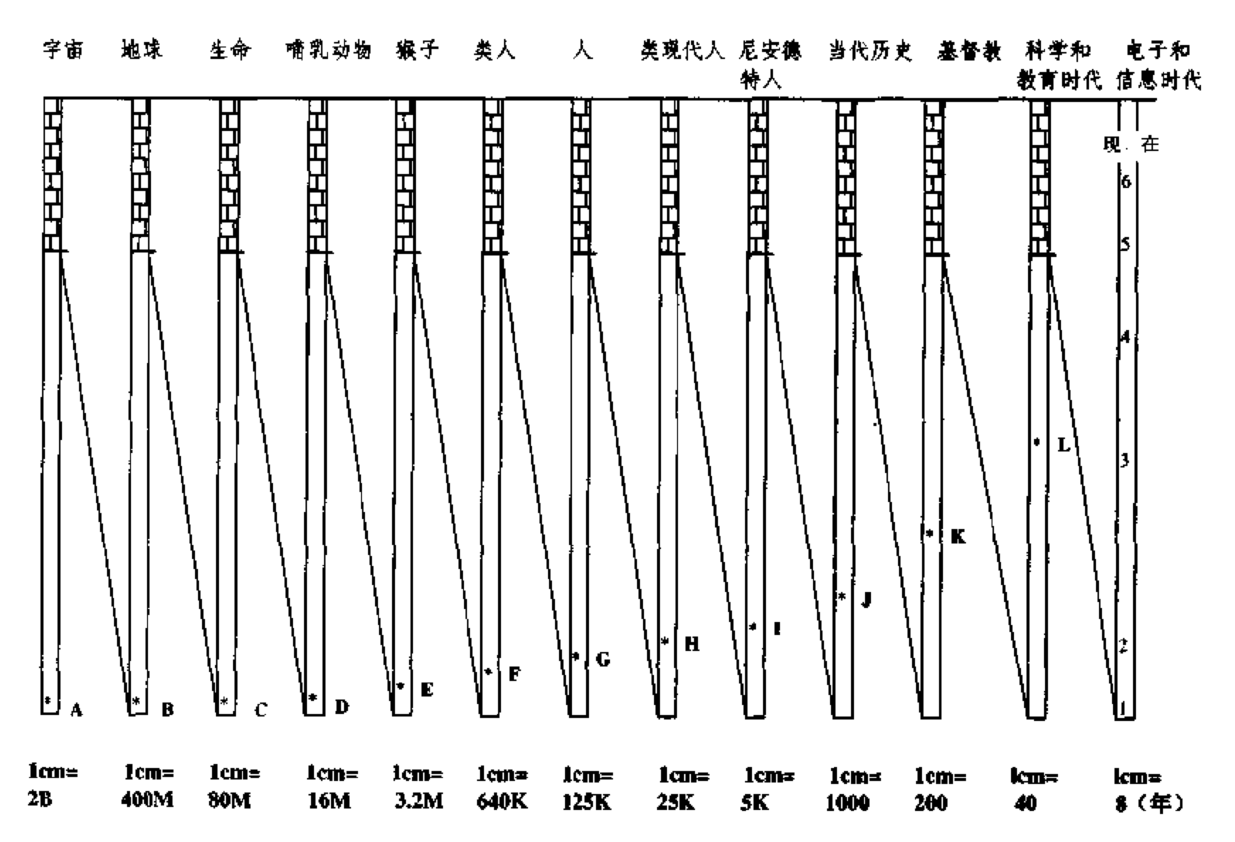
“事实上，欧洲的演化策略研究和美国的遗传算法研究已经在既不被接受也不被忽视的状态下生存了十多年了。另一个事实是，到目前为止，这两种思想以地理隔离的方式进化，且并没有产生组合的后代。现在是将大西洋两边的富含基因的思想源充分利用，并产生新一代算法的时候了，也是充分利用和炫耀大规模并行处理器系统的良好的环境的时候了”。

在本书的最后，我想再一次提一下本书的两个主要目标。第一是想让读者相信：进化是一个强有力的且一般化的概念，它应该在许多解决问题的技术中找到自身的位置。特别是，整个人工智能领域应该向演化技术靠拢，正如 Lawrence Fogel[125]在世界计算智能大会(World Congress on Computational Intelligence, Orlando, 27 June – 2 July 1994）中的权威发言中所说的：

“如果目标是产生人工智能，即以新方法解决新问题，那么使用任何固定的规则集合都是不合适的。规则要求解决每个问题时都应该简单地进行演化…”

第二个目标是强调各种演化技术之间的相似性，这些技术在本书中的讨论是从对特定问题的演化程序的构造展开的。而这些技术所有的不同，即遗传算法、演化策略、演

化规划、遗传规划等，都是处于较低水平上的，都属于某种技术细节。各种技术都是使用不同的染色体表达和适当的遗传算子集合。



*A 宇宙和第一个化学元素的出现
*B 地球上岩石的出现
*C 多细胞动物的起源
*D 胎盘动物的起源
*E 原上猿的起源
*F 南方古猿的起源
*F 直立人的起源
*G 类现代人的起源
*H 可罗马农人的起源
*I 第一个文明的起源
*J 文艺复兴的开始
*K 现代科学和生产技术的起源

1–第一个星球　2–量子力学理论　3–第一次原子弹爆炸；第一台计算机
4–人类第一次进入宇宙　5–VLSI 芯片　6–高温半导体

图 16.2 进化速度的超指数增长。图中左边一列表示跨距宇宙的起源到现在的整个时间段。第二列是对第一列部分的放大，它位于连接两者的对角线上，等等。



附录 A　一个简单实用的遗传算法 C 代码

这是一个非常简单的遗传算法源代码，是由 Denis Cormier（North Carolina State University）开发的，Sita S. Raghavan （University of North Carolina at Charlotte）修正。代码保证尽可能少，实际上也不必查错；在许多例子中，清楚的代价是效率。对一特定的应用修正此代码，用户只需改变常数的定义并且定义“评价函数”即可。注意代码的设计是求最大问题，其中的目标函数值只能取正值；且函数值和个体的适应值之间没有区别。该系统使用比率选择、精华模型、单点杂交和均匀变异。如果用 Gaussian 变异替换均匀变异，可能能得到更好的结果，因此我们鼓励读者对系统进行这样的改变——见附录 D 的练习 6。

代码并没有使用任何图形，甚至也没有屏幕输出，主要是保证在平台之间高度的可移植性：读者可以从 ftp.uncc.edu, 目录 coe/evol 中的文件 prog.c 中获得。

要求的输入文件应该被命名为 `gadata.txt'；系统产生的输出文件为 'galog.txt'。输入文件由几行组成：数目对应于变量数。且每一行提供次序——对应的变量上下边界。如，第一行为第一个变量提供上下边界，第二行为第二个变量提供上下边界，等等。

```
/***************************************************************************************************/
/* This is a simple genetic algorithm implementation where the                                     */
/* evaluation function takes positive values only and the                                          */
/* fitness of an individual is the same as the value of the                                        */
/* objective function                                                                              */
/***************************************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Change any of these parameters to match your needs */

#define POPSIZE 50                /* population size */
#define MAXGENS 1000              /* max. number of generations */
#define NVARS 3                   /* no. of problem variables */
#define PXOVER 0.8                /* probability of crossover */
#define PMUTATION 0.15            /* probability of mutation */
#define TRUE 1
#define FALSE 0

int generation;                   /* current generation no. */
int cur_best;                     /* best individual */
FILE *galog;                      /* an output file */
struct genotype                   /* genotype (GT), a member of the population */
{
  double gene[NVARS];             /* a string of variables */
  double fitness;                 /* GT's fitness */
  double upper[NVARS];            /* GT's variables upper bound */
  double lower[NVARS];            /* GT's variables lower bound */
  double rfitness;                /* relative fitness */
```

```
   double cfitness;                          /* cumulative fitness */
};
struct genotype population[POPSIZE+1];      /* population */
struct genotype newpopulation[POPSIZE+1];        /* new population; */
                                               /* replaces the */
                                               /* old generation */

/* Declaration of procedures used by this genetic algorithm */

void initialize(void);
double randval(double, double);
void evaluate(void);
void keep_the_best(void);
void elitist(void);
void select(void);
void crossover(void);
void Xover(int,int);
void swap(double *, double *);
void mutate(void);
void report(void);

/***************************************************************************************************/
/* Initialization function: Initializes the values of genes                                       */
/* within the variables bounds. It also initializes (to zero)                                     */
/* all fitness values for each member of the population. It                                       */
/* reads upper and lower bounds of each variable from the                                         */
/* input file `gadata.txt'. It randomly generates values                                          */
/* between these bounds for each gene of each genotype in the                                     */
/* population. The format of the input file `gadata.txt' is                                       */
/* var1_lower_bound var1_upper bound                                                              */
/* var2_lower_bound var2_upper bound ...                                                          */
/***************************************************************************************************/
void initialize(void)
{
FILE *infile;
int i, j;
double lbound, ubound;

if ((infile = fopen("gadata.txt","r"))==NULL)
      {
      fprintf(galog,"\nCannot open input file!\n");
      exit(1);
      }

/* initialize variables within the bounds */

for (i = 0; i < NVARS; i++)
      {
      fscanf(infile, "%lf",&lbound);
      fscanf(infile, "%lf",&ubound);

      for (j = 0; j < POPSIZE; j++)
           {
           population[j].fitness = 0;
           population[j].rfitness = 0;
           population[j].cfitness = 0;
           population[j].lower[i] = lbound;
           population[j].upper[i]= ubound;
           population[j].gene[i] = randval(population[j].lower[i], population[j].upper[i]);
           }
```

```
        }

fclose(infile);
}

/***************************************************************************/
/* Random value generator: Generates a value within bounds                 */
/***************************************************************************/
double randval(double low, double high)
{
double val;
val = ((double)(rand()%1000)/1000.0)*(high - low) + low;
return(val);
}

/***************************************************************************/
/* Evaluation function: This takes a user defined function.                */
/* Each time this is changed, the code has to be recompiled.               */
/* The current function is:  x[1]^2-x[1]*x[2]+x[3]                         */
/***************************************************************************/
void evaluate(void)
{
int mem;
int i;
double x[NVARS+1];

for (mem = 0; mem < POPSIZE; mem++)
        {
        for (i = 0; i < NVARS; i++)
                x[i+1] = population[mem].gene[i];

        population[mem].fitness = (x[1]*x[1]) - (x[1]*x[2]) + x[3];
        }
}

/***************************************************************************/
/* Keep_the_best function: This function keeps track of the                */
/* best member of the population. Note that the last entry in              */
/* the array Population holds a copy of the best individual                */
/***************************************************************************/
void keep_the_best()
{
int mem;
int i;
cur_best = 0; /* stores the index of the best individual */

for (mem = 0; mem < POPSIZE; mem++)
        {
        if (population[mem].fitness > population[POPSIZE].fitness)
                {
                cur_best = mem;
                population[POPSIZE].fitness = population[mem].fitness;
                }
        }
/* once the best member in the population is found, copy the genes */
for (i = 0; i < NVARS; i++)
        population[POPSIZE].gene[i] = population[cur_best].gene[i];
}
```

```
/*********************************************************************************************************************/
/* Elitist function: The best member of the previous generation                                                      */
/* is stored as the last in the array. If the best member of                                                         */
/* the current generation is worse then the best member of the                                                       */
/* previous generation, the latter one would replace the worst                                                       */
/* member of the current population                                                                                  */
/*********************************************************************************************************************/
void elitist()
{
int i;
double best, worst;              /* best and worst fitness values    */
int best_mem, worst_mem;         /* indexes of the best and worst member */

best = population[0].fitness;
worst = population[0].fitness;
for (i = 0; i < POPSIZE - 1; ++i)
      {
      if(population[i].fitness > population[i+1].fitness)
            {
            if (population[i].fitness> = best)
                  {
                  best = population[i].fitness;
                  best_mem = i;
                  }
            if (population[i+1].fitness <= worst)
                  {
                  worst = population[i+1].fitness;
                  worst_mem = i + 1;
                  }
            }

        else
            {
            if (population[i].fitness < = worst)
                  {
                  worst = population[i].fitness;
                  worst_mem = i;
                  }
            if (population[i+1].fitness >= best)
                  {
                  best = population[i+1].fitness;
                  best_mem = i + 1;
                  }
            }
      }

/* if best individual from the new population is better than */
/* the best individual from the previous population, then    */
/* copy the best from the new population; else replace the   */
/* worst individual from the current population with the     */
/* best one from the previous generation                     */

if (best >= population[POPSIZE].fitness)
    {
    for (i = 0; i < NVARS; i++)
        population[POPSIZE].gene[i] = population[best_mem].gene[i];
    population[POPSIZE].fitness = population[best_mem].fitness;
    }
else
    {
    for (i = 0; i < NVARS; i++)
```

```
            population[worst_mem].gene[i] = population[POPSIZE].gene[i];
        population[worst_mem].fitness = population[POPSIZE].fitness;
        }
}

/**************************************************************************/
/* Selection function: Standard proportional selection for               */
/* maximization problems incorporating elitist model - makes             */
/* sure that the best member survives                                    */
/**************************************************************************/
void select(void)
{
int mem, i, j, k;
double sum = 0;
double p;

/* find total fitness of the population */
for (mem = 0; mem < POPSIZE; mem++)
        {
        sum += population[mem].fitness;
        }
/* calculate relative fitness */
for (mem = 0; mem < POPSIZE; mem++)
        {
        population[mem].rfitness =   population[mem].fitness/sum;
        }
population[0].cfitness = population[0].rfitness;

/* calculate cumulative fitness */
for (mem = 1; mem < POPSIZE; mem++)
        {
        population[mem].cfitness =   population[mem-1].cfitness +
                              population[mem].rfitness;
        }

/* finally select survivors using cumulative fitness. */

for (i = 0; i < POPSIZE; i++)
        {
        p = rand()%1000/1000.0;
        if (p < population[0].cfitness)
                newpopulation[i] = population[0];
        else
                {
                for (j = 0; j < POPSIZE;j++)
                        if (p >= population[j].cfitness &&
                                    p<population[j+1].cfitness)
                              newpopulation[i] = population[j+1];
                }
        }
/* once a new population is created, copy it back */

for (i = 0; i < POPSIZE; i++)
        population[i] = newpopulation[i];
}
/**************************************************************************/
/* Crossover selection: selects two parents that take part in            */
/* the crossover. Implements a single point crossover                    */
/**************************************************************************/
void crossover(void)
```

```
{
int i, mem, one;
int first   =   0;              /* count of the number of members chosen */
double x;

for (mem = 0; mem < POPSIZE; ++mem)
      {
      x = rand()%1000/1000.0;
      if (x < PXOVER)
            {
            ++first;
            if (first % 2 == 0)
                  Xover(one, mem);
            else
                  one = mem;
            }
      }
}
/***************************************************************************/
/* Crossover: performs crossover of the two selected parents.              */
/***************************************************************************/
void Xover(int one, int two)
{
int i;
int point;                      /* crossover point */

/* select crossover point */
if(NVARS > 1)
   {
   if(NVARS == 2)
         point = 1;
   else
         point = (rand() % (NVARS - 1)) + 1;

   for (i = 0; i < point; i++)
        swap(&population[one].gene[i], &population[two].gene[i]);
   }
}

/***************************************************************************/
/* Swap: A swap procedure that helps in swapping 2 variables               */
/***************************************************************************/
void swap(double *x, double *y)
{
double temp;

temp = *x;
*x = *y;
*y = temp;

}

/***************************************************************************/
/* Mutation: Random uniform mutation. A variable selected for              */
/* mutation is replaced by a random value between lower and                */
/* upper bounds of this variable                                           */
/***************************************************************************/
void mutate(void)
{
int i, j;
```

```
double lbound, hbound;
double x;

for (i = 0; i < POPSIZE; i++)
        for (j = 0; j < NVARS; j++)
                {
                x = rand()%1000/1000.0;
                if (x < PMUTATION)
                        {
                        /* find the bounds on the variable to be mutated */
                        lbound = population[i].lower[j];
                        hbound = population[i].upper[j];
                        population[i].gene[j] = randval(lbound, hbound);
                        }
                }
}


/***************************************************************************************************************/
/* Report function: Reports progress of the simulation. Data                                                   */
/* dumped into the   output file are separated by commas                                                      */
/***************************************************************************************************************/
void report(void)
{
int i;
double best_val;                  /* best population fitness */
double avg;                        /* avg population fitness */
double stddev;                    /* std. deviation of population fitness */
double sum_square;                /* sum of square for std. calc */
double square_sum;                /* square of sum for std. calc */
double sum;               /* total population fitness */

sum = 0.0;
sum_square = 0.0;

for (i = 0; i < POPSIZE; i++)
        {
        sum += population[i].fitness;
        sum_square += population[i].fitness * population[i].fitness;
        }

avg = sum/(double)POPSIZE;
square_sum = avg * avg * (double)POPSIZE;
stddev = sqrt((sum_square - square_sum)/(POPSIZE - 1));
best_val = population[POPSIZE].fitness;

fprintf(galog, "\n%5d,         %6.3f, %6.3f, %6.3f \n\n", generation,
                                        best_val, avg, stddev);
}

/***************************************************************************************************************/
/* Main function: Each generation involves selecting the best                                                   */
/* members, performing crossover & mutation and then                                                           */
/* evaluating the resulting population, until the terminating                                                  */
/* condition is satisfied                                                                                       */
/***************************************************************************************************************/
void main(void)
{
int i;
if ((galog = fopen("galog.txt","w"))==NULL)
        {
```

```
        exit(1);
        }
generation = 0;

fprintf(galog, "\n generation   best   average   standard \n");
fprintf(galog, " number        value fitness   deviation \n");

initialize();
evaluate();
keep_the_best();
while(generation<MAXGENS)
        {
        generation++;
        select();
        crossover();
        mutate();
        report();
        evaluate();
        elitist();
        }
fprintf(galog,"\n\n Simulation completed\n");
fprintf(galog,"\n Best member: \n");

for (i = 0; i < NVARS; i++)
    {
    fprintf (galog,"\n var(%d) = %3.3f",i,population[POPSIZE].gene[i]);
    }
fprintf(galog,"\n\n Best fitness = %3.3f",population[POPSIZE].fitness);
fclose(galog);
printf("Success\n");
}

/***************************************************************************************************************************/
```

附录B 测试函数

一些可以用在各种实验的测试函数如下，其他的一些问题，可以参考[350]、[186]和[113]的附录 A[①]：

(1) De Jong 函数 F1：

$$\sum_{i=1}^{3} x_i^2, \qquad \text{其中 } -5.12 \le x_i \le 5.12$$

函数在$(x_1, x_2, x_3) = (0, 0, 0)$处有一个全局最小值 0。

(2) De Jong 函数 F2：

$$100(x_1^2 - x_2)^2 + (1-x_1)^2, \qquad \text{其中 } -2.048 \le x_i \le 2.048$$

函数在$(x_1, x_2) = (1, 1)$处有一个全局最小值 0。

(3) De Jong 函数 F3：

$$\sum_{i=1}^{5} \text{integer}(x_i), \qquad \text{其中 } -5.12 \le x_i \le 5.12$$

函数在$-5.12 \le x_i < -5.0$ 里有一个全局最小值 –30。

(4) De Jong 函数 F4：

$$\sum_{i=1}^{30} i\, x_i^4 + \text{Gauss}(0,1), \qquad \text{其中 } -1.28 \le x_i \le 1.28$$

函数（没有 Gaussian 噪声）在$(x_1, x_2, \ldots, x\{30\}) = (0,0,\ldots,0)$处有一个全局最小值 0。

(5) De Jong 函数 F5：

$$\frac{1}{1/K + \sum_{j=1}^{25} f_j^{-1}(x_1, x_2)}, \qquad \text{其中 } f_j(x_1, x_2) = c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6,$$

$-65.536 \le x_i \le 65.536$, K=500, $c_j = j$，且

$$[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \ldots & 32 & 32 & 32 \end{bmatrix}$$

函数在$(x_1, x_2) = (-32, -32)$处有一个全局最小值 0.998 。

(6) Schaffer 函数 F6：

---

① 也可参考[401]对构造测试函数的讨论。

$$0.5+\frac{\sin^2\sqrt{x_1^2+x_2^2}-0.5}{[1.0+0.001(x_1^2+x_2^2)]^2},\qquad 其中\ -100\le x_i\le 100$$

函数在$(x_1, x_2) = (0,0)$处有一个全局最小值 0。

(7) Schaffer 函数 F7：

$$(x_1^2+x_2^2)^{0.25}[\sin^2(50(x_1^2+x_2^2)^{0.1})+1.0],\qquad 其中\ -100\le x_i\le 100$$

函数在$(x_1, x_2) = (0,0)$有一个全局最小值 0。

(8) Goldstein–Price 函数：

$$[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]\cdot[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)],\qquad 其中 -2\le x_i\le 2$$

函数在$(x_1, x_2) = (0, -1)$有一个全局最小值 3。

(9) Branin RCOS 函数：

$$a(x_2-bx_1^2+cx_1-d)^2+e(1-f)\cos(x_1)+e,\qquad 其中\ -5\le x_1\le 10, 0\le x_2\le 15,$$

且 a=1，b=5.1/ (4π^2)，c=5/π，d=6，e=10，f =1/ (8π)。函数在三个不同的点：$(x_1, x_2) = (-\pi, 12.275), (\pi, 2.275)$ 和(9.42478, 2.475)处有一个全局最小值 0.397887。

(10) Shekel SQRN5, SQRN7, SQRN10 族 4 维函数：

$$s_3(x_1,x_2,x_3,x_4)=-\sum_{j=1}^{5}\frac{1}{\sum_{i=1}^{4}(x_i-a_{ij})^2+c_j}$$

$$s_4(x_1,x_2,x_3,x_4)=-\sum_{j=1}^{7}\frac{1}{\sum_{i=1}^{4}(x_i-a_{ij})^2+c_j}$$

$$s_5(x_1,x_2,x_3,x_4)=-\sum_{j=1}^{10}\frac{1}{\sum_{i=1}^{4}(x_i-a_{ij})^2+c_j}$$

其中 $0\le x_i\le 10$，这里 i=1,2,3,4，a_{ij}和 c_j的值列在表 1 中。

表 1 函数 s_3，s_4和 s_5数据

j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	c_j
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.6
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

这些函数在点$(x_1,x_2,x_3,x_4) = (4,4,4,4)$处有一全局最小值，值为：$s_{3,min} = -10.15320$，$s_{4,min} = -10.402820$ 及 $s_{5,min} = -10.53628$。

(11) 六峰驼返回函数（six-hump camel back function）：

$$(4-2.1x_1^2+\frac{1}{3}x_1^4)x_1^2+x_1x_2+(-4+4x_2^2)x_2^2,$$ 其中$-3\le x_1\le 3$及$-2\le x_2\le 2$。

函数在两个不同的点：$(x_1,x_2)=(-0.0898,0.7126)$ 和 $(0.0898,-0.7126)$处有一个全局最小值 -1.0316 。

(12) Shubert 函数：

$$\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\cdot\sum_{i=1}^{5} i\cos[(i+1)x_2+i],$$ 其中 $-10\le x_i\le 10$ 当 $i=1,2$

函数有 760 个局部最小，其中的 18 个是全局最小，其值为 -186.73。

(13) Stuckman 函数：

$$f8(x_1,x_2)=\begin{cases}\lfloor(\lfloor m_1\rfloor+1/2)\sin(a_1)/a_1\rfloor & 若\ 0\le x_1\le b\\ \lfloor(\lfloor m_2\rfloor+1/2)\sin(a_2)/a_2\rfloor & 若\ b<x_1\le 10\end{cases}$$

其中对 $i=1,2$，$0\le x_i\le 10$ ，m_i 是在 0 和 100 之间的一随机变量（$i=1,2$），b 是一在 0 和 10 之间的随机变量，且：

$$a_i=\lfloor\,|x_1-r_{1i}|\,\rfloor+\lfloor\,|x_2-r_{2i}|\,\rfloor$$

其中 r_{11} 是在 0 和 b 之间的一个随机变量，r_{12} 是在 b 和 10 之间的一个随机变量，r_{21} 是一个在 0 和 10 之间的随机变量，及 r_{22} 是一个在 0 和 10 之间的随机变量（所有的随机变量都是唯一的）。全局最大值位于

$$(x_1,x_2)=\begin{cases}(r_{11},r_{21}) & 当\ m_1>m_2\\ (r_{12},r_{22}) & 否则\end{cases}$$

(14) Easom 函数：

$$-\cos(x_1)\cos(x_2)\ e^{-(x_1-\pi)^2-(x_2-\pi)^2},$$ 其中对 $i=1,2$，$-100\le x_i\le 100$

函数在 $(x_1,x_2)=(\pi,\pi)$ 处有一个全局最小值 -1。

(15) Bohachevsky 函数 #1：

$$x_1^2+2x_2^2-0.3\cos(3\pi x_1)-0.4\cos(4\pi x_2)+0.7,$$ 其中 $-50\le x_i\le 50$

函数在$(x_1,x_2)=(0,0)$处有一个全局最小值 0。

(16) Bohachevsky 函数 #2：

$$x_1^2+2x_2^2-0.3\cos(3\pi x_1)\cos(4\pi x_2)+0.3,$$ 其中 $-50\le x_i\le 50$

函数在$(x_1,x_2)=(0,0)$处有一个全局最小值 0 。

(17) Bohachevsky 函数#3：

$$x_1^2+2x_2^2-0.3\cos(3\pi x_1+4\pi x_2)+0.3,$$ 其中 $-50\le x_i\le 50$

函数在$(x_1,x_2)=(0,0)$处有一个全局最小值 0。

(18) Colville 函数：

$$100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2+$$
$$10.1((x_2-1)^2+(x_4-1)^2)+19.8(x_2-1)(x_4-1),$$

其中 $-10\le x_i\le 10$。函数在 $(x_1,x_2,x_3,x_4)=(1,1,1,1)$处有一全局最小值 0。

附录 C　用于约束优化的测试函数

有几个测试函数可以用来实验各种约束优化。其他问题可见[350]、[186]和[113]中的附录 A：

(1) 问题[114]为求最小：

$$G1(x,y) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i^2$$

服从约束：

$$2x_1 + 2x_2 + y_6 + y_7 \le 10, \quad 2x_1 + 2x_3 + y_6 + y_8 \le 10$$
$$2x_2 + 2x_3 + y_7 + y_8 \le 10, \quad -8x_1 + y_6 \le 0$$
$$-8x_2 + y_7 \le 0, \quad -8x_3 + y_8 \le 0$$
$$-2x_4 - y_1 + y_6 \le 0, \quad -2y_2 - y_3 + y_7 \le 0$$
$$-2y_4 - y_5 + y_8 \le 0, \quad 0 \le x_i \le 1, \ i = 1,2,3,4,$$
$$0 \le y_i \le 1, \ i = 1,2,3,4,5,9, \ 0 \le y_i, \quad i = 6,7,8.$$

全局解是$(x^*, y^*) = (1,1,1,1,1,1,1,1,1,3,3,3,1)$及 $G1(x^*, y^*) = -15$。

(2) 问题[186]为求函数最小：

$$G2(\overline{X}) = x_1 + x_2 + x_3$$

其中

$$1 - 0.0025(x_4 + x_6) \ge 0, \quad x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \ge 0,$$
$$1 - 0.0025(x_5 + x_7 - x_4) \ge 0, \quad x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \ge 0,$$
$$1 - 0.01(x_8 - x_5) \ge 0, \quad x_3x_8 - 1250000 - x_3x_5 + 2500x_5 \ge 0,$$
$$100 \le x_1 \le 10000, \quad 1000 \le x_i \le 10000, \ i = 2, 3,$$

问题有 3 个线性和 3 个非线性约束。函数 $G2$ 是线性的，全局最小位于

$$\overline{X}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$$

其中 $G2(\overline{X}^*) = 7049.330923$。所有的六个约束在全局最优值都是处于激活态。

(3) 问题[186]是求函数的最小：

$$G3(\overline{X}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - x_8,$$

其中

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \ge 0,$$
$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \ge 0,$$
$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \ge 0,$$
$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \ge 0$$
$$-10.0 \le x_i \le 10.0, \quad i = 1, \ldots, 7.$$

问题有 4 个非线性约束；函数 $G3$ 是非线性的，全局最小值位于

$$\overline{X}^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$$

其中 $G3(\overline{X}^*) = 680.6300573$。四个约束中的两个在全局最优解处被激活（第一个和最后一个）。

(4) 问题[186]是求一个函数的最小：

$$G4(\overline{X}) = e^{x_1x_2x_3x_4x_5}$$

服从约束 $x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10$, $x_2x_3 - 5x_4x_5 = 0$, $x_1^3 + x_2^3 = -1$,

$-2.3 \le x_i \le 2.3$, $i=1,2$, $-3.2 \le x_i \le 3.2$, $i=3,4,5$

问题有 3 个非线性等式：非线性函数 $G4$ 的全局最优解位于：

$$\overline{X}^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.7636450),$$

其中 $G4(\overline{X}^*) = 0.0539498478$。

(5) 问题[186]是求一个函数的最小：

$$G5(\overline{X}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 + 2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45,$$

其中 $105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \ge 0$,

$-3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 + 7x_4 + 120 \ge 0$,

$-10x_1 + 8x_2 + 17x_7 - 2x_8 \ge 0$,

$-x_1^2 - 2(x_2-2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \ge 0$,

$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \ge 0$,

$-5x_1^2 - 8x_2 - (x_3-6)^2 + 2x_4 + 40 \ge 0$,

$3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \ge 0$,

$-0.5(x_1-8)^2 - 2(x_2-4)^2 - 3x_5^2 + x_6 + 30 \ge 0$,

$-10.0 \le x_i \le 10.0$, $i = 1, \ldots, 10$.

问题有 3 个线性和 5 个非线性约束：函数 $G5$ 是二次型的，全局最小值位于

$$\overline{X}^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927),$$

其中 $G5(\overline{X}^*) = 24.3062091$。八个约束中有六个在全局最优点处于激活态（除了最后两个）。

(6) 问题[220]是求一个函数的最大：

$$G6(x) = \left| \frac{\sum_{i=1}^{n}\cos^4(x_i) - 2\prod_{i=1}^{n}\cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

其中 $\prod_{i=1}^{n} x_i > 0.75$, $\sum_{i=1}^{n} x_i < 7.5n$，对 $1 \le i \le n$, $1 < x_i < 10$

问题有 2 个非线性约束；函数 $G6$ 是非线性的，且全局最大值是未知的。一些较好的解（由 Genocop III 发现，见第 7 章）如下。对 $n = 20$：

x = (3.16311359, 3.13150430, 3.09515858, 3.06016588, 3.03103566,
2.99158549, 2.95802593, 2.92285895, 0.48684388, 0.47732279,
0.48044473, 0.48790911, 0.48450437, 0.44807032, 0.46877760,
0.45648506, 0.44762608, 0.44913986, 0.44390863, 0.45149332)

相应的 $G6(x)$ = 0.80351067。对 n =50：

x = (6.28006029, 3.16155291, 3.15453815, 3.14085174, 3.12882447,
3.11211085, 3.10170507, 3.08703685, 3.07571769, 3.06122732,
3.05010581, 3.03667951, 3.02333045, 3.00721049, 2.99492717,
2.97988462, 2.96637058, 2.95589066, 2.94427204, 2.92796040,
0.40970641, 2.90670991, 0.46131119, 0.48193336, 0.46776962,
0.43887550, 0.45181099, 0.44652876, 0.43348753, 0.44577143,
0.42379948, 0.45858049, 0.42931050, 0.42928645, 0.42943302,
0.43294361, 0.42663351, 0.43437257, 0.42542559, 0.41594154,
0.43248957, 0.39134723, 0.42628688, 0.42774364, 0.41886297,
0.42107263, 0.41215360, 0.41809589, 0.41626775, 0.42316407),

相应的 $G6(x)$= 0.8331。

(7) 问题[186]为求最小：

$$G7(\boldsymbol{x}, y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$$

服从约束： $6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \le 6.5$, $10x_1 + 10x_3 + y \le 20$,

$0 \le x_i \le 1$, $0 \le y$

全局解为 $(\boldsymbol{x}^*, y^*) = (0,1,0,1,1,20)$ 及 $G7(\boldsymbol{x}^*, y^*) = -213$。

(8) 问题[186]为求最小：

$$G8(\boldsymbol{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \ldots + x_{10}}\right)$$

其中

$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2$, $x_4 + 2x_5 + x_6 + x_7 = 1$

$x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1$, $x_i \ge 0.000001$ $(i=1,\ldots,10)$

其中

$c_1 = -6.089$; $c_2 = -17.164$; $c_3 = -34.054$; $c_4 = -5.914$; $c_5 = -24.721$;

$c_6 = -14.986$; $c_7 = -24.100$; $c_8 = -10.708$; $c_9 = -26.662$; $c_{10} = -22.179$;

由 GENOCOP（第 7 章）发现的最好解是：

x^*=(.04034785, .15386976, .77497089, .00167479, .48468539,
.00068965, .02826479, .01849179, .03849563, .10128126),

相应的目标函数值等于 −47.760765。

(9) 问题[113]为求最大：

$$G9(x)=\frac{3x_1+x_2-2x_3+0.8}{2x_1-x_2+x_3}+\frac{4x_1-2x_2+x_3}{7x_1+3x_2-x_3}$$

服从约束：　$x_1+x_2-x_3\le 1$,　　$-x_1+x_2-x_3\le -1$,

$12x_1+5x_2+12x_3\le 34.8$,　$12x_1+12x_2+7x_3\le 29.1$,

$-6x_1+x_2+x_3\le -4.1$,　　$0\le x_i$, $i=1,2,3$.

全局解为 $x^*=(1,0,0)$，及 $G9(x^*)=2.471428$。

(10) 问题[114]是求最大：

$$G10(x)=x_1^{0.6}+x_2^{0.6}-6x_1-4x_3+3x_4,$$

服从约束：

$-3x_1+x_2-3x_3=0$,　　$x_1+2x_3\le 4$,

$x_2+2x_4\le 4$,　　$x_1\le 3$,

$x_4\le 1$,　　$0\le x_i$,　$i=1,2,3,4$

知道的最好全局解为 $x^*=(4/3, 4, 0, 0)$ 及 $G10(x^*)=-4.5142$。

(11) 问题[114]是求最小：

$$G11(x,y)=6.5x-0.5x^2-y_1-2y_2-3y_3-2y_4-y_5,$$

服从约束：

$x+2y_1+8y_2+y_3+3y_4+5y_5\le 16$,

$-8x-4y_1-2y_2+2y_3+4y_4-y_5\le -1$,

$2x+0.5y_1+0.2y_2-3y_3-y_4-4y_5\le 24$,

$0.2x+2y_1+0.1y_2-4y_3+2y_4+2y_5\le 12$,

$-0.1x-0.5y_1+2y_2+5y_3-5y_4+3y_5\le 3$,

$y_3\le 1$, $y_4\le 1$,　$y_5\le 2$,

$x\ge 0$, $y_i\ge 0$, 其中 $1\le i\le 5$

全局解为 $(x, y^*)=(0, 6, 0, 1, 1, 0)$，及 $G11(x, y^*)=-11$。

(12) 问题是按照下面的方式由三个分离的问题[186]构造：求最小

$$G12(x)=\begin{cases} f_1=x_2+10^{-5}(x_2-x_1)^2-1.0 & 若\ 0\le x_1<2 \\ f_2=\frac{1}{27\sqrt{3}}((x_1-3)^2-9)x_2^3 & 若\ 2\le x_1<4 \\ f_3=\frac{1}{3}(x_1-2)^3+x_2-\frac{11}{3} & 若\ 4\le x_1\le 6 \end{cases}$$

服从约束：　$x_1/\sqrt{3}-x_2\ge 0$,

$-x_1-\sqrt{3}x_2+6\ge 0$,

$0\le x_1\le 6$,　$x_2\ge 0$。

函数 $G12$ 有三个全局解：

$x_1^*=(0,0)$,　$x_2^*=(3, \sqrt{3})$和 $x_3^*=(4,0)$,

在所有的情况下，$G12(x_i^*)=-1$ $(i=1,2,3)$。

附录D　演化计算方法课程安排

讲授一个演化计算技术的课程的最好方法可能是将它组织成一个面向方案（project）的课程。一些初步进程的实现，如一个简单的遗传算法，可以使学生抓住演化技术的基本概念，这可在学期的前几周完成，这门课是很具挑战性的。

可以尝试几个可能的实验方案；总之，本文应该是提出问题而不是回答问题！最后一章“结论”给出了当前研究领域的方向；几个方案可以从那里获得。当然，方案在复杂性和完成它们所需要的时间是可以变化的；它们中的一些可以十分简单，其他一些可能要求几个学生组成一个小组。无论如何，记住：方案表包含一个对可能问题的随意收集，它可能也应该引发其他思想。

所以，在本附录里，我们提供各种方案的一个表及几个对报告的计算实验的评注 —— 这通常发生在一些结果十分有趣且值得公布时！

D.1　几个可能的方案

(1) 在几个测试函数上比较算法：爬山法、随机爬山法、模拟退火、遗传算法、进化策略。

(2) 在几个测试例子上比较遗传算法的几种选择方案：比例、加权、竞争。

(3) 在几个测试例子上比较不同染色体编码的遗传算法的 3 个版本（二进制、灰码和浮点数）。

(4) 对一个典型约束问题，如背包问题，实验不同的约束处理方法，解码器，修补算法，罚函数等。

(5) 比较不同的浮点表达式算子（如 Gaussian 变异对非均匀变异、实验启发式规则等）。

(6) 采用附录 A 的简单编码，并将它用于数值优化问题。这可以以多种方式实现：

① 提供较好的输入文件，其中用户可以定义变量数及其他参数；

② 修正系统以处理最小化问题；

③ 修正系统以处理目标函数的所有值（正或负），而不仅仅是正值；

④ 用 Gauss 变异替换均匀变异（大约的随机数 Q 遵从预测值 μ和变化值σ^2 的正则分布，可以从区间 [0..1]产生 12 个随机数 R_i, i=1,2,..., 12（均匀分布）；然后 $Q=\mu+\sigma\left(\sum_{i=1}^{12} R_i-6\right)$ ）；

⑤ 用算术杂交替换单点杂交；

⑥ 引入其他算子，包括多亲体杂交，即计算几个亲体的“中心”作为规模坐标的评价值，并朝此中心移动最弱的个体；

⑦ 引入各种类型的变量，布尔、整数等；

⑧ 等等。

(7) 用浮点表达式和适当的算子比较进化策略和遗传算法。

(8) 设计并实验对一些演化方法参数的适应机制（算子概率、群体规模、变异步骤、杂交类型等等）。

(9) 采用当前的公共软件（如 GENOCOP）并用它处理整数和布尔变量。实验整数编程问题。

(10) 引入和实验一些非标准的遗传算法特征（性别、家族关系、个体的合作、配对产生对子体）。开展对这些启发式规则的证明，并将它实验到一些测试问题上。要十分小心可能的改进和增加系统复杂性之间的交换。

(11) 考虑目标函数随时间变化的问题。实验单倍体对双倍体染色体结构。

(12) 开发对一些演化系统的图形接口以显示当前搜索的统计结果。

(13) 从你专长的领域取一些非常规的优化问题（数据库、操作研究、图像处理、工程设计、模糊控制、人工神经网络、游戏、机器人等等）。开发这类问题的演化应用程序。将它和对此问题知名的方法进行比较（见下一部分）。

D.2 有关启发式方法计算实验报告的评述

许多演化技术是用来对几个测试问题的实验进行评价的，如附录 B 和附录 C 所列的例子；通常很难归纳这些实验结果，以便对一特定技术作出一些“全局的”看法。但是，通过将一个方法和其他完整建立的技术进行比较，有可能证明此方法对几个精心挑选的问题的实用性。跟从[26]，一个新的启发式方法的优点可能包括下面几点：

- 它产生高质量解的速度快于其他方法；
- 它比其他方法更容易识别高质量解；
- 对问题特征、数据质量或调节参数的差异，它比其他方法更不敏感；
- 它更容易实现；
- 它能应用于较宽范围的问题。

另一段引证[26]是：

“如果一个启发式方法的研究报告是揭示型的——通过建立对一个算法的执行结果的推理和其行为的解释来提供对样本启发式设计或者问题结构的洞察，而

且是理论型的——提供理论洞察，如有关解的质量的跳跃——则它们是很有价值。"

Barr 等[26]的文献给出了有关如何用启发式方法设计和报告计算实验极佳的综述材料。例如，在预备和报告你的实验时，遵从[26]中的五步可能是你非常应该做的：

- 定义实验目标；
- 选择对执行和探究因素的度量；
- 设计并执行实验；
- 分析数据并抽取结论；
- 报告实验结果。

这些问题都是很重要的。例如，实验目标可能是变化的：正如[26]中所描述的：

"对算法的计算实验通常承担着（a）比较对同一类问题不同算法的执行效果，或者（b）特征化或者描述一个算法单独执行的结果。当这些目标有些相互关联时，应该特别识别测试所要完成的是什么。即，要回答的是什么问题，要测试的是什么假设。"

同时，你可能选择最好解的质量作为执行结果的度量，或者是达到它所耗费的时间，或者是算法达到"可接受"解所耗费的时间，或者是方法的力度，并列出几种可能。在多数情况下，有必要将新方法和已建立的技术对一类给定问题进行比较。应该记住分析主要因素，像问题的大小对解的质量和计算量的影响。最终的报告也应该包含能使读者再生结果的所有信息。

在万维网（World Wide Web）上；有许多可用的标准的测试问题库，这些应该是被任何实验研究所经常使用的，有关操作研究测试问题集的信息，见 OR-Library (or.library@ic.ac.uk)；可用 ftp 访问 mscmga.ms.ic.ac.uk ，或访问 WWW 站点：http://mscmga.ms.ic.ac.uk 。



参考文献

[1] Aarts, E.H.L. and Korst, J., *Simulated Annealing and Boltzmann Machines*, John Wiley, Chichester, UK, 1989.

[2] Aarts, E.H.L. and Lenstra,J.K., (Editors), *Local Search in Combinatorial Optimization*, John Wiley, Chichester, UK, 1995.

[3] Abuali, F.N., Wainwright, R.L., Schoenefeld, D.A., *Determinant Factorization: A New Encoding Scheme for SpanningTrees Applied to the Probabilistic Minimum Spanning Tree Problem*, in [103], pp.470–477.

[4] Ackley, D.H., *An Empirical Study of Bit Vector Function Optimization*, in [73], pp.170–204.

[5] Adler, D., *Genetic Algorithms and Simulated Annealing: A Marriage Proposal*, Proceedings of the IEEE International Conference on Neural Networks, March 28–April 1, 1993, pp.1104–1109.

[6] Akl, S.G., *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[7] Alander, J.T., *Proceedings of the First Finnish Workshop on Genetic Algorithms and their Applications*, Helsinki University of Technology, Finland, November 4–5, 1992.

[8] Angeline, P.J. and Kinnear, K.E. (Editors), *Advances in Genetic Programming II*, MIT Press, Cambridge, MA, 1996.

[9] Antonisse, H.J., *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint*, in [344], pp.86–91.

[10] Antonisse, H.J. and Keller, K.S.,*Genetic Operators for High Level Knowledge Representation*,in [171], pp.69-76.

[11] Arabas, J., Mulawka, J., and Pokras niewicz, J., *A New Class of the Crossover Operators for the Numerical Optimization*, in [103], pp.42-48.

[12] Arabas, J.,Michalewicz, Z., and Mulawka, J.,*GAVaPS——a Genetic Algorithm with Varying Population Size*,in[275],pp.73-78.

[13] Attia, N.F., *New Methods of Constrained Optimization using Penalty Functions*, Ph.D. Thesis, Essex University, England,1985.

[14] Axelrod, R., *Genetic Algorithm for the Prisoner Dilemma Problem*, in [73], pp.32–41.

[15] Bäck , T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1995.

[16] Bäck , T., and Hoffmeister, F., *Extended Selection Mechanisms in Genetic Algorithms*, in [32], pp.92–99.

[17] Bäck , T., Fogel, D.B., and Michalewicz, Z.,*Handbook of Evolutionary Computation*, University Oxford Press,NewYork,1996.

[18] Bäck , T., Hoffmeister, F., and Schwefel, H.-P., *A Survey of Evolution Strategies*, in [32], pp.2–9.

[19] Bäck , T., Rudolph, G., and Schwefel, H.-P., *Evolutionary Programming and Evolution Strategies: Similarities and Differences*, in [124], pp.11–22.

[20] Bäck , T. and Schwefel, H.-P., *An Overview of Evolutionary Algorithms for Parameter Optimization*, Evolutionary Computation, Vol.1, No.1, pp.1–23, 1993.

[21] Bagchi, S., Uckun, S., Miyabe, Y., and Kawamura, K., *Exploring Problem-Specific Recombination Operatorsfor Job Shop Scheduling*, in [32], pp.10–17.

[22] Baker, J.E., *Adaptive Selection Methods for Genetic Algorithms*, in [167], pp.101–111.

[23] Baker, J.E., *Reducing Bias and Inefficiency in the Selection Algorithm*, in [171], pp.14–21.

[24] Bala, J., De Jong, K.A., Pachowicz, P., *Using Genetic Algorithms to Improve the Performance ofClassification Rules Produced by Symbolic Inductive Method*, Proceedings of the 6th International Symposium on Methodologies of Intelligent Systems, Springer-Verlag, Lecture Notes in Artificial Intelligence, Vol.542, pp.286–295, 1991.

[25] Banach, S., *Sur les operations dans les ensembles abstraits et leur applications aux equations integrales*, Fundamenta Mathematica, Vol.3, pp.133–181, 1922.

[26] Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart, W.R.,*Designing and Reporting on Computational Experiments withHeuristic Methods*, Proceedings of the International Conference on Metaheuristics for Optimization, Kluwer Publishing, pp.1–17, 1995.

[27] Bean, J.C. and Hadj-Alouane, A.B., *A Dual Genetic Algorithm for Bounded Integer Programs*, Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53, 1992.

[28] Beasley, D., Bull, D.R., and Martin, R.R., *An Overview of Genetic Algorithms: Part 1, Foundations*, University Computing, Vol.15, No.2, pp.58–69, 1993.

[29] Beasley, D., Bull, D.R., and Martin, R.R., *An Overview of Genetic Algorithms: Part 2, Research Topics*, University Computing, Vol.15, No.4, pp.170–181, 1993.

[30] Beasley, D., Bull, D.R., and Martin, R.R., *A Sequential Niche Technique for Multimodal Function Optimization*, Evolutionary Computation, Vol.1, No.2, pp.101–125, 1993.

[31] Beasley, J.E. and Chu, P.C., *A Genetic Algorithm for the Set Covering Problem*, Technical Report, The management School, Imperial College, July 1994.

[32] Belew, R. and Booker, L. (Editors), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[33] Bellman, R., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[34] Bennett, K., Ferris, M.C., and Ioannidis, Y.E., *A Genetic Algorithm for Database Query Optimization*, in [32], pp.400–407.

[35] Bersini, H. and Varela, F.J., *The Immune Recruitment Mechanism: A Selective Evolutionary Strategy*, in [32], pp.520–526.

[36] Bertoni, A. and Dorigo, M., *Implicit Parallelism in Genetic Algorithms*, Artificial Intelligence, Vol.61, no.2, pp.307–314, 1993.

[37] Bertsekas, D.P., *Dynamic Programming, Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[38] Bethke, A.D., *Genetic Algorithms as Function Optimizers*, Doctoral Dissertation, University of Michigan, 1980.

[39] Betts, J.T., *An Accelarated Multiplier Method for Nonlinear Programming*, Journal of Optimization Theory and Applications, Vol.21, No.2, pp.137–174, 1977.

[40] Biggs, M.C., *Constrained Minimization Using Recursive Quadratic Programming: Some Alternative Subproblem Formulations*, Towards Global Optimization (Eds. L.C.W. Dixon and G.P. Szego), North-Holland, 1975.

[41] Biggs, M.C., *A Numerical Comparison Between Two Approaches to the Nonlinear Programming Problem*, Numerical Optimization Centre Technical Report No. 77, The Hatfield Polytechnic, 1976.

[42] Bilchev, G. and Parmee, I.C., *Ant Colony Search vs. Genetic Algorithms*, Technical Report, Plymouth Engineering Design Centre, University of Plymouth, 1995.
[43] Bilchev, G., *Private communication*, April 1995.
[44] Bland, R.G. and Shallcross, D.F.,*Large Traveling Salesman Problems Arising From Experiments inX-Ray Crystallography: A Preliminary Report on Computation*, Operations Research Letters, Vol.8, pp.125–128, 1989.
[45] Booker, L.B., *Intelligent Behavior as an Adaptation to the Task Environment*, Doctoral Dissertation, University of Michigan, 1982.
[46] Booker, L.B., *Improving Search in Genetic Algorithms*, in [73], pp.61–73.
[47] Bosworth, J., Foo, N., and Zeigler, B.P., *Comparison of Genetic Algorithms with Conjugate GradientMethods*, Washington, DC, NASA (CR–2093), 1972.
[48] Bowen, J. and Dozier, G., *Solving Constraint Satisfaction Problems Using a Genetic/Systematic Search Hybrid that Realizes when to Quit*, in[103], pp.122–129.
[49] Brindle, A., *Genetic Algorithms for Function Optimization*, Doctoral Dissertation, University of Alberta, Edmonton, 1981.
[50] Brooke, A., Kendrick, D., and Meeraus, A., *GAMS: A User's Guide*, The Scientific Press, Belmont, CA, 1988.
[51] Broyden, C.G. and Attia, N.F., *A Smooth Sequential Penalty Function Method for Solving Nonlinear Programming Problem*, Lecture Notes in Control and Information Science 59, System Modelling and Optimization, Proceedings of the 11th IFIP Conference, July 1983, Springer-Verlag, 1983.
[52] Broyden, C.G. and Attia, N.F., *Penalty Functions, Newton's Method, and Quadratic Programming*, Journal of Optimization Theory and Applications, Vol.58, No.3., 1988.
[53] Bruns, R., *Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling*, in [129], pp.352–359.
[54] Bruns, R., *Scheduling*, in [17], section F1.5.
[55] Bui, T.N. and Eppley, P.H., *A Hybrid Genetic Algorithm for the Maximum Clique Problem*, in [103], pp.478–483.
[56] Bui, T.N. and Moon, B.R., *On Multi-Dimensional Encoding/Crossover*, in [103], pp.49–56.
[57] Burke, E.K., Elliman, D.G., and Weare, R.F., *A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems*, in [103], pp.605–610.
[58] Carey, M.R. and Johnson, D.S., *Computers and Intractibility – A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.
[59] Cartwright, H.M., and Mott, G.F., *Looking Around: Using Clues from the Data Space to Guide Genetic Algorithm Searches*, in[32], pp.108–114.
[60] Cavaretta, M.J., *Using Cultural Algorithm to Control Genetic Operators*, in [378], pp.158–166.
[61] Cetverikov, S.S., *On Some Aspects of the Evolutionary Process From the Viewpoint of Modern Genetics*(in Russian), Journal Exper. Biol., Vol.2, No.1, pp.3–54, 1926.
[62] Cleveland, G.A. and Smith, S.F., *Using Genetic Algorithms to Schedule Flow Shop Releases*, in [344], pp.160–169.
[63] Colorni, A., Dorigo, M., and Maniezzo, V., *Genetic Algorithms and Highly Constrained Problems: The Time-Table Case*, in [351], pp.55–59.
[64] Colville, A.R., *A Comparative Study on Nonlinear Programming Codes*, IBM Scientific Center Report 320-2949, New York, 1968.
[65] Coombs, S., and Davis, L., *Genetic Algorithms and Communication Link Speed Design:Theoretical Considerations*, in [171],pp.252–256.
[66] Cooper, L., and Steinberg, D., *Introduction to Methods of Optimization*, W.B. Saunders, London, 1970.
[67] Cormier, D.R., *Pluto: A General Purpose Evolution Programming Shell*, Technical Report, August 17, 1993, Department of Industrial Engineering, North Carolina State University.
[68] Craighurst, R. and Martin, W., *Enhancing GA Preformance through Crossover Prohibitions Based on Ancestry*, in [103], pp.130–135.
[69] Davidor, Y., *Genetic Algorithms and Robotics*, World Scientific, 1991.
[70] Davidor, Y., Schwefel, H.-P., and Männer, R. (Editors), Proceedings of the Third International Conference on Parallel ProblemSolving from Nature (PPSN), Springer-Verlag, New York, 1994.
[71] Davis, L., *Applying Adaptive Algorithms to Epistatic Domains*, Proceedings of the International Joint Conference on Artificial Intelligence, pp.162–164, 1985.
[72] Davis, L., *Job Shop Scheduling with Genetic Algorithms*,in [167], pp.136–140.
[73] Davis, L., (Editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, San Mateo, CA, 1987.
[74] Davis, L. and Steenstrup, M., *Genetic Algorithms and Simulated Annealing: An Overview*, in [73], pp.1–11.
[75] Davis, L. and Ritter, F., *Schedule Optimization with Probabilistic Search*, Proceedings of the Third Conference on Artificial Intelligence Applications, Computer Society Press, pp.231–236, 1987.
[76] Davis, L., and Coombs, S., *Genetic Algorithms and Communication Link Speed Design:Constraints and Operators*, in [171], pp.257–260.
[77] Davis, L., *Adapting Operator Probabilities in Genetic Algorithms*, in [344], pp.61–69.
[78] Davis, L., (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
[79] Davis, L., *Bit-Climbing, Representational Bias, and Test Suite Design*, in [32], pp.18–23.
[80] Davis, T.E. and Principe, J.C., \em *A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm*, in [32], pp.174–181.
[81] Dawkins, R., *The Selfish Gene*, Oxford University Press, New York, 1976.
[82] De Jong, K.A., "*An Analysis of the Behavior of a Class of Genetic Adaptive Systems*", (Doctoral dissertation, University of Michigan),*Dissertation Abstract International*, 36(10), 5140B, (University Microfilms No 76-9381).
[83] De Jong, K.A., *Adaptive System Design: A Genetic Approach*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.10, No.3,pp.556–574, 1980.

[84] De Jong, K.A., *Genetic Algorithms: A 10 Year Perspective*, in [167], pp.169–177.
[85] De Jong, K.A., *On Using Genetic Algorithms to Search Program Spaces*, in [171], pp.210–216.
[86] De Jong, K.A., *Learning with Genetic Algorithm: An Overview*, Machine Learning, Vol.3, pp.121–138, 1988.
[87] De Jong, K.A., (Editor), *Evolutionary Computation*, MIT Press, 1993.
[88] De Jong K.A., *Genetic-Algorithm-Based Learning*, in [226], pp.611–638.
[89] De Jong, K., *Genetic Algorithms: A 25 Year Perspective*, in [415],pp.125–134.
[90] De Jong K.A., Spears, W.M., *Using Genetic Algorithms to Solve NP-Complete Problems*, in [344], pp.124–132.
[91] De Jong K.A., Spears, W.M., *Using Genetic Algorithms for Supervised Concept Learning*, Proceedings of the Second International Conference on Tools for AI, pp.335–341, 1990.
[92] De La Maza, M. and Yuret, D., *Dynamic Hill Climbing*, AI Expert, March 1994, pp.26–31.
[93] Dhar, V. and Ranganathan, N., *Integer Programming vs. Expert Systems: An Experimental Comparison*, Communications of the ACM, Vol.33, No.3, pp.323–336, 1990.
[94] Dixmier, J., *General Topology*, Springer-Verlag,1984.
[95] Dozier, G., Bahler, D., and Bowen, J., *Solving Small and Large Scale Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm*, in [275],pp.306–311.
[96] Eades, P. and Lin, X., *How to Draw a Directed Graph*, Technical Report, Department of Computer Science, University of Queensland,Australia, 1989.
[97] Eades, P. and Tamassia, R., *Algorithms for Drawing Graphs: An Annotated Bibliography*,Technical Report No. CS-89-09, Brown University, Department of Computer Science, October, 1989.
[98] Eiben, A.E., Aarts, E.H.L., Van Hee, K.M., *Global Convergence of Genetic Algorithms: On Infinite Markov Chain Analysis*, in [351], pp.4–12.
[99] Eiben, A.E., Raue, P.-E., and Ruttkay, Zs., *Solving Constraint Satisfaction Problems Using GeneticAlgorithms*, in [276], pp.542–547.
[100] Eiben, A.E., Raue, P.-E., and Ruttkay, Zs., *Genetic Algorithms with Multi-parent Recombination*, in [70], pp.78–87.
[101] Esbensen, H., *Finding (Near-)Optimal Steiner Trees in Large Graphs*,in [103], pp.485–491.
[102] Eshelman, L.J., *The CHC Adaptive Search Algorithm: How to Have Safe Search WhenEngaging in Nontraditional Genetic Recombination*, in [318], pp.265–283.
[103] Eshelman, L.J., (Editor),Proceedings of the Sixth International Conference on GeneticAlgorithms, Morgan Kaufmann, San Mateo, CA, 1995.
[104] Eshelman, L.J., Caruana, R.A., and Schaffer, J.D., *Biases in the Crossover Landscape*,in [344], pp.10–19.
[105] Eshelman, L.J., and Schaffer, J.D., *Preventing Premature Convergence in Genetic Algorithms byPreventing Incest*, in [32], pp.115–122.
[106] Even, S., Itai, A., and Shamir, A., *On the Complexity of Timetable and Multicommodity Flow Problems*,SIAM Journal on Computing, Vol.5, No.4, 1976, pp.691–703.
[107] Evolutionary Computation, Vol.2, No.1, Spring 1994; special issue on classifier systems.
[108] Falkenauer, E., *The Grouping Genetic Algorithms - Widening the Scope of the GAs*, Belgian Journal of Operations Research,Statistics and Computer Science, Vol.33, 1993, pp.79–102.
[109] Falkenauer, E., *A New Representation and Operators for GAs Applied to Grouping Problems*, Evolutionary Computation, Vol.2,No.2, pp.123–144.
[110] Falkenauer, E., *Solving Equal Piles with a Grouping Genetic Algorithm*, in [103], pp.492–497.
[111] Fiacco, A.V., and McCormick, G.P., *Nonlinear Programming*, John Wiley, Chichester, UK, 1968.
[112] Fletcher, R., *Practical Methods of Optimization*, Vol.2, *of Constrained Optimization*, John Wiley, Chichester, UK, 1981.
[113] Floudas, C.A. and Pardalos, P.M., *A Collection of Test Problems for Constrained Global OptimizationAlgorithms*, Springer-Verlag, Lecture Notes in Computer Science,Vol.455, 1987.
[114] Floudas, C.A. and Pardalos, P.M., *Recent Advances in Global Optimization*, Princeton Series inComputer Science, Princeton University Press, Princeton, NJ, 1992.
[115] Fogarty, T.C., *Varying the Probability of Mutation in the Genetic Algorithm*, in[344], pp.104–109.
[116] Fogel, D.B., *An Evolutionary Approach to the Traveling Salesman Problem*,Biol. Cybern., Vol.60, pp.139–144, 1988.
[117] Fogel, D.B., *Evolving Artificial Intelligence*, PhD Thesis, University of California, San Diego, 1992.
[118] Fogel, D.B. (Editor), IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.
[119] Fogel, D.B., *An Introduction to Simulated Evolutionary Optimization*, IEEE Transactions on Neural Networks, special issue on EP, Vol.5, No.1, 1994.
[120] Fogel, D.B., *Evolving Behaviours in the Iterated Prisoner's Dilemma*,Evolutionary Computation, Vol.1, No.1, pp.77–97, 1993.
[121] Fogel, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 1995.
[122] Fogel, D.B. and Atmar, J.W., *Comparing Genetic Operators with Gaussian Mutation in SimulatedEvolutionary Process Using Linear Systems*, Biol. Cybern., Vol.63, pp.111-114, 1990.
[123] Fogel, D.B. and Atmar, W., *Proceedings of the First Annual Conference on EvolutionaryProgramming*, La Jolla, CA, 1992, Evolutionary Programming Society.
[124] Fogel, D.B. and Atmar, W., *Proceedings of the Second Annual Conference on EvolutionaryProgramming*, La Jolla, CA, 1993, Evolutionary Programming Society.
[125] Fogel, L.J., *Evolutionary Programming in Perspective: The Top-Down View*,in [415], pp.135–146.
[126] Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.
[127] Fonseca, C.M. and Fleming, P.J., *An Overview of Evolutionary Algorithms in MultiobjectiveOptimization*, Evolutionary Computation, Vol.3, No.1, 1995, pp.165–180.
[128] Forrest, S., *Implementing Semantic Networks Structures Using the ClassifierSystem*, in [167], pp.24–44.
[129] Forrest, S. (Editor),Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo,

CA,1993.

[130] Foux, G., Heymann, M., Bruckstein, A., *Two-Dimensional Robot Navigation Among Unknown Stationary Polygonal Obstacles*. IEEE Transactions on Robotics and Automation. Vol.9, pp.96–102, 1993.

[131] Fox, B.R., and McMahon, M.B., *Genetic Operators for Sequencing Problems*, in [318], pp.284–300.

[132] Fox, M.S., *Constraint–Directed Search: A Case Study of Job–Shop Scheduling*, Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[133] Freville, A., and Plateau, G., *Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems*. European Journal of Operational Research, Vol. 24, pp.206–215.

[134] Garey, M. and Johnson, D., *Computers and Intractability*, W.H. Freeman, San Francisco, 1979.

[135] Gen, M., Wasserman, G.S., and Smith, A.E.,(Guest Editors), Computers and Industrial Engineering Journal, special issue on genetic algorithms and industrial engineering, Vol.30, No.2, 1996.

[136] Gendreau, M., Hertz, A., and Laporte, G., *A Tabu Search Heuristic for Vehicle Routing*, CRT-7777, Centre de Recherche sur les transports, Universite de Montreal, 1991.

[137] Gill, P.E., Murray, W., and Wright, M.H., *Practical Optimization*, Academic Press, London, 1978.

[138] Gillies, A.M., *Machine Learning Procedures for Generating Image Domain Feature Detectors*, Doctoral Dissertation, University of Michigan, 1985.

[139] Giordana, A. and Saitta, L., *REGAL: An Integrated System for Learning Relations Using GeneticAlgorithms*, Proceedings of the International Workshop on Multistrategy Learning, MSL-93, Harpers Ferry, VA, pp.234–249, 1993.

[140] Giordana, A., Saitta, L., and Zini, F., *Learning Disjunctive Concepts with Distributed Genetic Algorithms*,in [275], pp.115–119.

[141] Glover, D.E., *Solving a Complex Keyboard Configuration Problem ThroughGeneralized Adaptive Search*, in [73], pp.12–27.

[142] Glover, F., *Heuristics for Integer Programming Using Surrogate Constraints*, Decision Sciences, Vol.8, No.1, pp.156–166, 1977.

[143] Glover, F., *Tabu Search — Part I*, ORSA Journal on Computing, Vol.1, No.3, pp.190–206, 1989.

[144] Glover, F., *Tabu Search — Part II*, ORSA Journal on Computing, Vol.2, No.1, pp.4–32, 1990.

[145] Glover, F., *Tabu Search for Nonlinear and Parametric Optimization*, Technical Report, Graduate School of Business, University of Colorado at Boulder; preliminary version presented at the EPFL Seminar on OR and AI Search Methods for Optimization Problems, Lausanne, Switzerland, November 1990.

[146] Glover, F., *Genetic Algorithms and Scatter Search: Unsuspected Potentials*, Statistics and Computing, Vol.4, pp.131–140.

[147] Glover, F., *Tabu Search Fundamentals and Uses*, Graduate School of Business, University of Colorado, 1995.

[148] Glover, F. and Kochenberger, G., *Critical Event Tabu Search for Multidimensional Knapsack Problems*, Proceedings of the InternationalConference on Metaheuristics for Optimization, Kluwer Publishing, pp.113–133, 1995.

[149] Goldberg, D.E., *Genetic Algorithm and Rule Learning in Dynamic Control System*, in [167], pp.8–15.

[150] Goldberg, D.E., *Dynamic System Control Using Rule Learning and Genetic Algorithms*, in Proceedings of the International Joint Conference on Artificial Intelligence, 9, pp.588–592, 1985.

[151] Goldberg, D.E., *Optimal Initial Population Size for Binary-Coded Genetic Algorithms*, TCGA Report No.85001, Tuscaloosa, University of Alabama, 1985.

[152] Goldberg, D.E., *Simple Genetic Algorithma and the Minimal,Deceptive Problem*, in [73], pp.74–88.

[153] Goldberg, D.E., *Sizing Populations for Serial and Parallel Genetic Algorithms*, in [344], pp.70–79.

[154] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[155] Goldberg, D.E., *Messy Genetic Algorithms: Motivation, Analysis, and First Results*, Complex Systems, Vol.3, pp.493–530, 1989.

[156] Goldberg, D.E., *Zen and the Art of Genetic Algorithms*, in [344], pp.80–85.

[157] Goldberg, D.E., *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking*, University of Illinois at Urbana-Champaign, Technical Report No. 90001, September 1990.

[158] Goldberg, D.E., *Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale*, Complex Systems, Vol.4, pp.415–444, 1990.

[159] Goldberg, D.E., Deb, K., and Korb, B., *Do not Worry, Be Messy*, in [32], pp.24–30.

[160] Gorges-Schleuter, M., *ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy*, in [167], pp.422–427.

[161] Goldberg, D.E., Milan, K., and Tidd, C., *Genetic Algorithms: A Bibligraphy*, IlliGAL Technical Report 92008, 1992.

[162] Goldberg, D.E. and Richardson, J., *Genetic Algorithms with Sharing for Multimodal Function Optimization*, in [171], pp.41-49.

[163] Goldberg, D.E. and Segrest, P., *Finite Markov Chain Analysis of Genetic Algorithms*, in [171], pp.1-8.

[164] Gorges-Schleuter, M., *ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy*, in [344], pp.422-427.

[165] Greene, F., *A Method for Utilizing Diploid/Dominance in Genetic Search*, in [275], pp.439–444.

[166] Grefenstette, J.J., *GENESIS: A System for Using Genetic Search Procedures*, Proceedings of the 1984 Conference on Intelligent Systems and Machines, pp.161–165.

[167] Grefenstette, J.J., (Editor),*Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[168] Grefenstette, J.J., Gopal, R., Rosmaita, B., and Van Gucht, D., *Genetic Algorithm for the TSP*, in [167], pp.160–168.

[169] Grefenstette, J.J., *Optimization of Control Parameters for Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 16, No.1, pp.122–128, 1986.

[170] Grefenstette, J.J., *Incorporating Problem Specific Knowledge into Genetic Algorithms*, in [73], pp.42–60.

[171] Grefenstette, J.J., (Editor), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[172] Gregory, J.W., *Nonlinear Programming FAQ*, 1995, Usenet sci.answers; available via anonymous ftp from rtfm.mit.edu in /pub/usenet/sci.answers/nonlinear-programming-faq.

[173] Groves, L., Michalewicz, Z., Elia, P., Janikow, C., *Genetic Algorithms for Drawing Directed Graphs*, Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, North-Holland, Amsterdam, pp.268–276, 1990.

[174] Hadj-Alouane, A.B. and Bean, J.C., *A Genetic Algorithm for the Multiple-Choice Integer Program*, Department of Industrial and Operations Engineering, The University of Michigan, TR 92-50, 1992.

[175] Han, S.P., *Superlinearly Convergent Variable Metric Algorithm for General Nonlinear Programming Problems*, Mathematical Programming, Vol.11, pp.263–282, 1976.

[176] Handley, S., *The Genetic Planner: The Automatic Generation of Plans for a Mobile Robot via Genetic Programming*, Proceedings of 8th IEEE International Symposium on Intelligent Control, August 25-27, 1993.

[177] Heitkötter, J., (Editor), *The Hitch-Hiker's Guide to Evolutionary Computation*, FAQ in comp.ai.genetic, issue 1.10, 20 December 1993.

[178] Herdy, M., *Application of the Evolution Strategy to Discrete Optimization Problems*, in [351], pp.188–192.

[179] Held, M. and Karp, R.M., *The Traveling Salesman Problem and Minimum Spanning Trees*, Operations Research, Vol.18, pp.1138–1162.

[180] Held, M. and Karp, R.M., *The Traveling Salesman Problem and Minimum Spanning Trees: Part II*, Mathematical Programming, Vol.1, pp.6–25.

[181] Hesser, J., Männer, R., and Stucky, O., *Optimization of Steiner Trees Using Genetic Algorithms*, in [344], pp.231–236.

[182] Hillier, F.S. and Lieberman, G.J., *Introduction to Operations Research*, Holden–Day, San Francisco, CA, 1967.

[183] Hinterding, R., *Mapping, Order-independent Genes and the Knapsack Problem*, in[275], pp.13–17.

[184] Hinterding, R., Gielewski, H., and Peachey, T.C., *The Nature of Mutation in Genetic Algorithms*, in [103], pp.65–72.

[185] Hobbs, M.F., *Genetic Algorithms, Annealing, and Dimension Alleles*, MSc. Thesis, Victoria University of Wellington, 1991.

[186] Hock, W. and Schittkowski K., *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, Vol.187, Springer-Verlag, 1981.

[187] Hoffmeister, F. and B®ck, T., *Genetic Algorithms and Evolution Strategies*, in [351], pp.455–470.

[188] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[189] Holland, J.H. and Reitman, J.S., *Cognitive Systems Based on Adaptive Algorithms*, in D.A. Waterman and F. Hayes-Roth (Editors), *Pattern-Directed Inference Systems*, Academic Press, New York,1978.

[190] Holland, J.H., *Properties of the Bucket Brigade*, in [167], pp.1–7.

[191] Holland, J.H., *Escaping Brittleness*, in R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Editors), *Machine Learning II*, Morgan Kaufmann Publishers, San Mateo, CA, 1986.

[192] Holland, J.H., *Royal Road Functions*, Genetic Algorithm Digest, Vol.7, No.22,12 August 1993.

[193] Holland, J.H., Holyoak, K.J., Nisbett, R.E., and Thagard, P.R., *Induction*, MIT Press, Cambridge, MA, 1986.

[194] Homaifar, A. and Guan, S., *A New Approach on the Traveling Salesman Problem by Genetic Algorithm*, Technical Report, North Carolina A & T State University, 1991.

[195] Homaifar, A., Lai, S. H.-Y., and Qi, X., *Constrained Optimization via Genetic Algorithms*, Simulation, Vol.62, 1994, pp.242–254.

[196] Horn, J. and Nafpliotis, N., *Multiobjective Optimization Using the Niched Pareto Genetic Algorithm*, Department of Computer Science, University of Illinois at Urbana-Champaign, IlliGAL Report 93005, 1993.

[197] Horowitz, E. and Sahni, S., *Fundamentals of Computer Algorithms*, Computer Science Press, Potomac, MD, 1978.

[198] Husbands, P., Mill, F., and Warrington, S., *Genetic Algorithms, Production Plan Optimization, and Scheduling*, in [351], pp.80–84.

[199] Ingber, L. and Rosen, B., *Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison*, Mathematical and Computer Modelling, Vol.16, No.11, pp.87–100, 1992.

[200] Janikow, C., *Inductive Learning of Decision Rules in Attribute-Based Examples: a Knowledge-Intensive Genetic Algorithm Approach*, PhD Dissertation, University of North Carolina at Chapel Hill, July 1991.

[201] Janikow, C., and Michalewicz, Z., *Specialized Genetic Algorithms for Numerical Optimization Problems*, Proceedings of the International Conference on Tools for AI, pp.798–804, 1990.

[202] Janikow, C., and Michalewicz, Z., *On the Convergence Problem in Genetic Algorithms*, UNCC Technical Report, 1990.

[203] Janikow, C. and Michalewicz, Z., *An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms*, in [32], pp.31–36.

[204] Jankowski, A., Michalewicz, Z., Ras, Z., and Shoff, D., *Issues on Evolution Programming*, in Computing and Information, (R. Janicki and W.W. Koczkodaj, Editors), North-Holland, Amsterdam, 1989, pp.459–463.

[205] Jarvis, R.A., and Byrne, J.C., *Robot Navigation: Touching, Seeing, and Knowing*, Proceedings of the 1st Australian Conference on AI, 1986.

[206] Jog, P., Suh, J.Y., Gucht, D.V., *The Effects of Population Size, Heuristic Crossover, and LocalImprovement on a Genetic Algorithm for the Traveling Salesman Problem*, in [344], pp.110–115.

[207] Johnson, D.S., *Local Optimization and the Traveling Salesman Problem*, in M.S. Paterson (Editor), Proceedings of the 17th Colloquium on Automata, Languages, and Programming, Springer-Verlag, Lecture Notes in Computer Science,Vol.443, pp.446–461, 1990.

[208] Johnson, D.S., *The Traveling Salesman Problem: A Case Study in Local Search*, presented during the Metaheuristics International Conference, Breckenridge, Colorado, July 22–26, 1995.

[209] Johnson, D.S., *Private communication*, October 1995.

[210] Joines, J.A. and Houck, C.R., *On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs*, in [276], pp.579–584.

[211] Jones, D.R. and Beltramo, M.A., *Solving Partitioning Problems with Genetic Algorithms*, in [32], pp.442–449.

[212] Jones, T., *A Description of Holland's Royal Road Function*, Evolutionary Computation, Vol.2, No.4, 1994, pp.409–415.

[213] Jones, T., *Crossover, Macromutation, and Population-based Search*, in [103], pp.73–80.

[214] Jones, T. and Forrest, S., *Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms*, in [103], pp.184–192.

[215] Juliff, K., *A Multi-chromosome Genetic Algorithm for Pallet Loading*, in [129], pp.467–473.

[216] Julstrom, B.A., *What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm*, in [103], pp.81–87.

[217] Kaufman, K.A., Michalski, R.S., and Scultz, A.C., *EMERALD 1: An Integrated System for Machine Learning and Discovery Programs for Education and Research*. Center for AI, George Mason University, User's Guide, No. MLI–89–12, 1989.

[218] Kambhampati, S.K., and Davis, L.S., *Multi-resolution Path Planning for Mobile Robots*, IEEE Journal of Robotics and Automation, RA-2, pp.135–145, 1986.

[219] Karp, R.M., *Probabilistic Analysis of Partitioning Algorithm for theTraveling Salesman Problem in the Plane*, Mathematics of Operations Research, Vol.2, No.3, 1977, pp.209–224.

[220] Keane, A., Genetic Algorithms Digest, Thursday, May 19, 1994, Volume 8, Issue 16.

[221] Kelly, J.P., Golden, B.L., and Assad, A.A., *Large Scale Controlled Rounding Using Tabu Search with Strategic Oscillation*, Annals of Operations Research, Vol.41, pp.69–84, 1993.

[222] Khatib O., *Real-Time Obstacles Avoidance for Manipulators and Mobile Robots*, International Journal of Robotics Research,Vol.5, pp.90–98, 1986.

[223] Khuri, S., Bäck , T., and Heitkötter, J., *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*, Proceedings of the ACM Symposium of Applied Computation (SAC '94).

[224] Kingdon, J., *Genetic Algorithms: Deception, Convergence and Starting Conditions*, Technical Report, Dept. of Computer Science, University College, London, 1992.

[225] Kinnear, K.E. (Editor), *Advances in Genetic Programming*, MIT Press, Cambridge, MA, 1994.

[226] Kodratoff, Y. and Michalski, R., *Machine Learning:An Artificial Intelligence Approach*, Vol.3, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[227] Korte, B., *Applications of Combinatorial Optimization*, talk at the 13th International Mathematical Programming Symposium, Tokyo, 1988.

[228] Koza, J.R., *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Report No. STAN–CS–90–1314, Stanford University, 1990.

[229] Koza, J.R., *A Hierarchical Approach to Learning the Boolean Multiplexer Function*, in [318], pp.171–192.

[230] Koza, J.R.,*Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm*, in [32], pp.37–44.

[231] Koza, J.R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[232] Koza, J.R., *Genetic Programming – 2*, MIT Press, Cambridge, MA, 1994.

[233] Laarhoven, P.J.M. van, and Aarts, E.H.L., *Simulated Annealing: Theory and Applications*, D. Reidel, Dordrecht, Holland, 1987.

[234] Langley, P., *On Machine Learning*, Machine Learning, Vol.1, No.1, pp.5–10, 1986.

[235] von Laszewski, G., *Intelligent Structural Operators for the k-Way Graph Partitioning Problem*, in [32], pp.45–52.

[236] Lawer, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., and Shmoys, D.B., *The Traveling Salesman Problem*, John Wiley, Chichester, UK, 1985.

[237] Lee, F.N., *The Application of Commitment Utilization Factor (CUF) to Thermal Unit Commitment*, IEEE Transactions on Power Systems, Vol.6, No.2, pp.691–698, May 1991.

[238] Le Riche, R., and Haftka, R.T., *Improved Genetic Algorithm forMinimum Thickness Composite Laminate Design*, Composites Engineering, Vol.3, No.1, pp.121-139, 1995.

[239] Le Riche, R., Knopf-Lenoir, C., and Haftka, R.T., *A Segregated Genetic Algorithm for Constrained Structural Optimization*, in [103], pp.558–565.

[240] Leler, W., *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, Reading, MA, 1988.

[241] Lidd, M.L., *Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms*, Technical Report, MITRE Corporation, 1991.

[242] Liepins, G.E., and Vose, M.D., *Representational Issues in Genetic Optimization*, Journal of Experimental and Theoretical Artificial Intelligence,Vol.2, No.2, pp.4–30, 1990.

[243] Lin, H.-S., Xiao, J., and Michalewicz, Z., *Evolutionary Algorithm for Path Planning in Mobile Robot Environment*, in [275], pp.211–216.

[244] Lin, H.-S., Xiao, J., and Michalewicz, Z., *Evolutionary Navigator for a Mobile Robot*, Proceedings of the IEEE Conference on Robotics and Automation, IEEE Computer Society Press, New York, 1994, pp.2199–2204.

[245] Lin, S. and Kernighan, B.W., *An Effective Heuristic Algorithm for the Traveling Salesman Problem*, Operations Research, pp.498–516, 1972.

[246] Litke, J.D., *An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes*, Communications of the ACM, Vol.27, No.12, pp.1227–1236, 1984.

[247] Logan, T.D., *GENOCOP: AN Evolution Program for Optimization Problems with Linear Constraints*, Master Thesis, Department of Computer Science, University of North Carolina at Charlotte, 1993.

[248] Lozano-Perez, T., and Wesley, M.A., *An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles*, Communications of the ACM, Vol.22, pp.560–570, 1979.

[249] Mahfoud, S.W., *A Comparison of Parallel and Sequential Niching Methods*, in [103], pp.136–143.

[250] Maniezzo, V., *Granularity Evolution*, Dipartimento di Elettronica e Informazione, Politecnico di Milano,Technical Report, 1993.

[251] Männer , R. and Manderick, B. (Editors), Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN), North-Holland, Elsevier Science Publishers, Amsterdam, 1992.

[252] Martello, S. and Toth, P., *Knapsack Problems*, John Wiley, Chichester, UK, 1990.

[253] McCormick, G.P., *Computability of Global Solutions to Factorable Nonconvex Programs*, Part I: Convex Underestimating Problems, Mathematical Programming, Vol.10, No.2 (1976), pp.147–175.

[254] McDonnell, J.R., Reynolds, R.G., and Fogel, D.B. (Editors), Proceedings of the Fourth Annual Conference on Evolutionary

Programming, The MIT Press, 1995.

[255] Messa, K. and Lybanon, M., *A New Technique for Curve Fitting*, Naval Oceanographic and Atmospheric Research Report JA 321:021:91, 1991.

[256] Michalewicz, Z. (Editor),Proceedings of the 5th International Conference on Statistical and Scientific Databases, Springer-Verlag, Lecture Notes in Computer Science, Vol.420, 1990.

[257] Michalewicz, Z., *A Genetic Algorithm for Statistical Database Security*, IEEE Bulletin on Database Engineering, Vol.13, No.3, September 1990, pp.19–26.

[258] Michalewicz, Z., *EVA Programming Environment*, UNCC Technical Report, 1990.

[259] Michalewicz, Z., *Optimization of Communication Networks*, Proceedings of the SPIE International Symposium on Optical Engineering and Photonics in Aerospace Sensing, SPIE, Bellingham, WA, 1991,pp.112–122.

[260] Michalewicz, Z. (Editor), *Statistical and Scientific Databases*, Ellis Horwood, London, 1991.

[261] Michalewicz, Z., *A Hierarchy of Evolution Programs: An Experimental Study*, Evolutionary Computation, Vol.1, No.1, 1993, pp.51–76.

[262] Michalewicz, Z., *Evolutionary Computation Techniques for Nonlinear Programming Problems*, International Transactions in Operational Research, 1994.

[263] Michalewicz, Z. (Ed.), Statistics & Computing, special issue on evolutionary computation, 1994.

[264] Michalewicz, Z., *A Hierarchy of Evolution Programs: An Experimental Study*, Evolutionary Computation, Vol.1, No.1, 1993, pp.51–76.

[265] Michalewicz, Z. (Editor),Statistics & Computing, special issue on evolutionary computation, 1994.

[266] Michalewicz, Z., *Genetic Algorithms, Numerical Optimization and Constraints*, in [103], pp.151–158.

[267] Michalewicz, Z., and Attia, N., *Evolutionary Optimization of Constrained Problems*, in [378], pp.98–108.

[268] Michalewicz, Z. and Janikow, C., *Genetic Algorithms for Numerical Optimization*, Statistics and Computing, Vol.1, No.1, 1991.

[269] Michalewicz, Z. and Janikow, C., *Handling Constraints in Genetic Algorithms*, in [32], pp.151–157.

[270] Michalewicz, Z. and Janikow, C., *GENOCOP: A Genetic Algorithm for Numerical Optimization Problemswith Linear Constraints*, accepted for publication,Communications of the ACM, 1992.

[271] Michalewicz, Z., Janikow, C., and Krawczyk, J., *A Modified Genetic Algorithm for Optimal Control Problems*, Computers & Mathematics with Applications, Vol.23, No.12, pp.83–94, 1992.

[272] Michalewicz, Z., Jankowski, A., Vignaux, G.A., *The Constraints Problem in Genetic Algorithms*, in *Methodologies of Intelligent Systems: Selected Papers*, M.L. Emrich, M.S. Phifer, B. Huber, M. Zemankova, Z. Ras (Editors),ICAIT, Knoxville, TN, pp.142–157, 1990.

[273] Michalewicz, Z., Krawczyk, J., Kazemi, M., Janikow, C., *Genetic Algorithms and Optimal Control Problems*, Proceedings of the 29th IEEE Conference on Decision and Control, Honolulu, pp.1664–1666, 1990.

[274] Michalewicz, Z., Logan, T.D., and Swaminathan, S., *Evolutionary Operators for Continuous Convex Parameter Spaces*, in [378], pp.84–97.

[275] Michalewicz, Z., Schaffer, D., Schwefel, H.-P., Fogel, D., Kitano, H. (Editors), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, IEEE Service Center, Piscataway, NJ, Volume 1,Orlando, 27–29 June, 1994.

[276] Michalewicz, Z., Schaffer, D., Schwefel, H.-P.,Fogel, D., Kitano, H. (Editors), *Proceedings of the First IEEEInternational Conference on Evolutionary Computation*, IEEE Service Center, Piscataway, NJ, Volume 2,Orlando, 27–29 June, 1994.

[277] Michalewicz, Z., Vignaux, G.A., Groves, L., *Genetic Algorithms for Optimization Problems*, Proceedings of the 11th NZ Computer Conference, Wellington, New Zealand, pp.211–223, 1989.

[278] Michalewicz, Z., Vignaux, G.A., Hobbs, M., *A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem*, ORSA Journal on Computing, Vol.3, No.4, pp.307–316, 1991.

[279] Michalewicz, Z. and Xiao, J., *Evaluation of Paths in Evolutionary Planner/Navigator*, Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 30–31, 1995, pp.45–52.

[280] Michalski, R., *A Theory and Methodology of Inductive Learning*, in R. Michalski, J. Carbonell, T. Mitchell (Editors), *Machine Learning:An Artificial Intelligence Approach*, Vol.1, Tioga Publishing Co., Palo Alto, CA, 1983, and Springer-Verlag, 1994,pp.83–134.

[281] Michalski, R., Mozetic, I., Hong, J., Lavrac, N., *The AQ15 Inductive Learning System: An Overview and Experiments*, Technical Report, Department of Computer Science, University of Illinois at Urbana–Champaign, 1986.

[282] Michalski, R. and Watanabe, L., *Constructive Closed–Loop Learning: Fundamental Ideas and Examples*, MLI–88, George Mason University, 1988.

[283] Michalski, R., *A Methodological Framework for Multistrategy Task–Adaptive Learning*, Methodologies for Intelligent Systems, Vol.5, in Z. Ras, M. Zemankowa, M. Emrich (Editors), North-Holland, Amsterdam, 1990.

[284] Michalski, R. and Kodratoff, Y., *Research in Machine Learning: Recent Progress, Classification of Methods, and Future Directions*, in [226], pp.3–30.

[285] Montana, D.J., and Davis, L., *Training Feedforward Neural Networks Using Genetic Algorithms*, Proceedings of the 1989 International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo,CA, 1989.

[286] Mühlenbein , H., *Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization*, in [344], pp.416-421.

[287] Mühlenbein , H., *Parallel Genetic Algorithms in Combinatorial Optimization*, in O. Balci, R. Sharda, and S.A. Zenios (Editors), *Computer Science and Operations Research—New Developments in Their Interfaces*, Pergamon Press, New York, 1992,pp.441–453.

[288] Mühlenbein , H., Gorges-Schleuter, M., Krämer , O., *Evolution Algorithms in Combinatorial Optimization*, Parallel Computing, Vol.7, pp.65–85, 1988.

[289] Mühlenbein , H. and Schlierkamp-Vosen, D., *Predictive Models for the Breeder Genetic Algorithm*, Evolutionary Computation, Vol.1, No.1, pp.25–49, 1993.

[290] Mühlenbein , H. and Voigt, H.-M., *Gene Poool Recombination for the Breeder Genetic Algorithm*, Proceedings of the

Metaheuristics International Conference, Breckenridge, Colorado, July 22–26, 1995. pp.19–25.

[291] Murray, W., *An Algorithm for Constrained Minimization*. Optimization, (Ed: R. Fletcher), pp.247–258, Academic Press, London, 1969.

[292] Murtagh, B.A. and Saunders, M.A., *MINOS 5.1 User's Guide*. Report SOL 83-20R, December 1983, revised January 1987, Stanford University.

[293] Muselli, M. and Ridella, S., *Global Optimization of Functions with the Interval genetic Algorithm*. Complex Systems, Vol.6, pp.193–212, 1992.

[294] Myung, H., Kim, J.-H., and Fogel, D.B., *Preliminary Investigations into a Two-Stage Method of Evolutionary Optimization on Constrained Problems*, in [254], pp.449–463.

[295] Nadhamuni, P.V.R., *Application of Co-evolutionary Genetic Algorithm to a Game*, Master Thesis, Department of Computer Science,University of North Carolina, Charlotte, 1995.

[296] von Neumann, J., *Theory of Self-Reproducing Automata*, edited by Burks, University of Illinois Press, 1966.

[297] Nissen, V., *Evolutionary Algorithms in Management Science: An Overview and List of References*, European Study Group for Evolutionary Economics, 1993.

[298] Ng, K.P, and Wong, K.C., *A New Diploid Scheme and Dominance Change Mechanism for Non-Stationary Function Optimization*, in [103], pp.159–166.

[299] Oliver, I.M., Smith, D.J., and Holland, J.R.C., *A Study of Permutation Crossover Operators on the Traveling Salesman Problem*, in [171], pp.224–230.

[300] Olsen, A., *Penalty Functions and the Knapsack Problem*, in [276], pp.554–558.

[301] Orvosh, D. and Davis, L., *Shall We Repair? Genetic Algorithms, Combinatorial Optimization,and Feasibility Constraints*, in [129], p.650.

[302] Padberg, M., and Rinaldi, G., *Optimization of a 532-City Symmetric Travelling Salesman Problem*,Technical Report IASI–CNR, Italy, 1986.

[303] Paechter, B., Luchian, H., and Petruic, M., *Two Solutions to the General Timetable Problem Using Evolutionary Methods*, in [275], pp.300–305.

[304] Palmer, C.C. and Kershenbaum, A., *Representing Trees in Genetic Algorithms*, in [275], pp.379–384.

[305] Pardalos, P., *On the Passage from Local to Global in Optimization*, in "Mathematical Programming", J.R. Birge and K.G. Murty(Editors), The University of Michigan, 1994.

[306] Paredis, J., *Exploiting Constraints as Background Knowledge for Genetic Algorithms: a Case-study for Scheduling*, in [251], pp.229–238.

[307] Paredis, J., *Genetic State-Space Search for Constrained Optimization Problems*, Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, 1993.

[308] Paredis, J., *Co-evolutionary Constraint Satisfaction*, in [70], pp.46–55.

[309] Paredis, J., *The Symbiotic Evolution of Solutions and their Representations*, in [103], pp.359–365.

[310] Pavlidid, T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

[311] Potter, M. and De Jong, K., *A Cooperative Coevolutionary Approach to Function Optimization*, in [70], pp.249–257.

[312] Powell, D. and Skolnick, M.M., *Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints*, in [129], pp.424–430.

[313] Powell, M.J.D., *Variable Metric Methods for Constrained Optimization*, Mathematical Programming: The State of the Art, (Eds: A.Bachem, M. Grötschel and B. Korte), pp.288–311, Springer-Verlag, 1983.

[314] Quinlann, J.R., *Induction of Decision Trees*, Machine Learning, Vol.1, No.1, 1986.

[315] Radcliffe, N.J., *Forma Analysis and Random Respectful Recombination*, in [32], pp 222–229.

[316] Radcliffe, N.J., *Genetic Set Recombination*, in [398], pp.203–219.

[317] Radcliffe, N.J., and George, F.A.W., *A Study in Set Recombination*, in [129], pp.23–30.

[318] Rawlins, G., *Foundations of Genetic Algorithms*, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems,Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[319] Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann–Holzboog Verlag, Stuttgart, 1973.

[320] Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, London, 1993.

[321] Reinelt, G., *TSPLIB – A Traveling Salesman Problem Library*, ORSA Journal on Computing, Vol.3, No 4, pp.376–384, 1991.

[322] Reinke, R.E. and Michalski, R., *Incremental Learning of Concept Descriptions*, in D. Michie (Editor), Machine Intelligence XI, 1985.

[323] Rendell, Larry A., *Genetic Plans and the Probabilistic Learning System: Synthesis and Results*, in [167], pp.60–73.

[324] Renders, J.-M., and Bersini, H., *Hybridizing Genetic Algorithms with Hill-climbing Methods for Global Optimization: Two Possible Ways*, in [275], pp.312–317.

[325] Reynolds, R.G., *An Introduction to Cultural Algorithms*, in [378], pp.131–139.

[326] Reynolds, R.G., Brown, W., and Abinoja, E.O., *Guiding Parallel Bi-Directional Search Using Cultural Algorithms*, in [378], pp.167–174.

[327] Reynolds, R.G. and Maletic, J.I., *The Use of Version Space Controlled Genetic Algorithms to Solvethe Boole Problem*, International Journal on Artificial Intelligence Tools, Vol.2, No.2, pp.219–234, 1993.

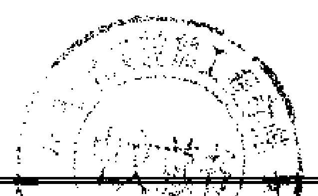
[328] Reynolds, R.G. and Maletic, J.I., *Learning to Cooperate Using Cultural Algorithms*, in [378], pp.140–149.

[329] Reynolds, R.G., Michalewicz, Z., and Cavaretta, M., *Using Cultural Algorithms for Constraint Handling in Genocop*, in [254], pp.289–305.

[330] Reynolds, R.G. and Sverdlik, W., *Solving Problems in Hierarchically Structured Systems Using Cultural Algorithms*, in [124], pp.144–153.

[331] Reynolds, R.G., Zannoni, E., and Posner, R.M., *Learning to Understand Software Using Cultural Algorithms*, in[378], pp.150–157.

[332] Richardson, J.T., Palmer, M.R., Liepins, G., and Hilliard, M., *Some Guidelines for Genetic Algorithms with Penalty Functions*, in [344], pp.191–197.
[333] Ronald, E., *When Selection Meets Seduction*, in [103], pp.167–173.
[334] Rudolph, G., *Convergence Analysis of Canonical Genetic Algorithms*, IEEE Transactions on Neural Networks, special issue on evolutionary computation, Vol.5, No.1, 1994.
[335] Rumelhart, D. and McClelland, J., *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol.1, MIT Press, Cambridge, MA, 1986.
[336] Saravanan, N. and Fogel, D.B., *A Bibliography of Evolutionary Computation & Applications*, Department of Mechanical Engineering, Florida Atlantic University, Technical Report No. FAU-ME-93-100, 1993.
[337] Sarle, W., *Kangaroos*, article posted on comp.ai.neural-nets on 1 September 1993.
[338] Sasieni, M., Yaspan, A., and Friedman, L., *Operations Research Methods and Problems*, John Wiley, Chichester, UK, 1959.
[339] Schaffer, J.D., *Some experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*, PhD Dissertation, Vanderbilt University, Nashville, 1984.
[340] Schaffer, J.D., *Learning Multiclass Pattern Discrimination*, in [167], pp.74–79.
[341] Schaffer, J.D. and Morishima, A., *An Adaptive Crossover Distribution Mechanism for Genetic Algorithms*, in [171], pp.36–40.
[342] Schaffer, J.D., *Some Effects of Selection Procedures on Hyperplane Sampling by Genetic Algorithms*, in [73], pp.89–103.
[343] Schaffer, J., Caruana, R., [103], L., and Das, R., *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, in [344], pp.51–60.
[344] Schaffer, J., (Editor), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 1989.
[345] Schaffer, J.D., Whitley, D., and [103], L.J., *Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art*, Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, MD, June 6, 1992, pp.1–37.
[346] Schoenauer, M., and Xanthakis, S., *Constrained GA Optimization*, in [129], pp.573–580.
[347] Schraudolph, N. and Belew, R., *Dynamic Parameter Encoding for Genetic Algorithms*, Machine Learning, Vol.9, No.1, pp.9–21, 1992.
[348] Schwefel, H.-P., *Numerical Optimization for Computer Models*, John Wiley, Chichester, UK, 1981.
[349] Schwefel, H.-P., *Evolution Strategies: A Family of Non-Linear Optimization Techniques Based on Imitating Some Principles of Organic Evolution*, Annals of Operations Research, Vol.1, pp.165–167, 1984.
[350] Schwefel, H.-P., *Evolution and Optimum Seeking*, John Wiley, Chichester, UK, 1995.
[351] Schwefel, H.-P. and Männer, R. (Editors), Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN), Springer-Verlag, Lecture Notes in Computer Science, Vol.496, 1991.
[352] Schwefel, H.-P., *Private communication*, July 1991.
[353] Seniw, D., *A Genetic Algorithm for the Traveling Salesman Problem*, MSc Thesis, University of North Carolina at Charlotte, 1991.
[354] Shaefer, C.G., *The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique*, in [171], pp.50–55.
[355] Shibata, T., and Fukuda, T., *Robot Motion Planning by Genetic Algorithm with Fuzzy Critic*, Proceedings of the 8th IEEE International Symposium on Intelligent Control, August 25–27, 1993.
[356] Shing, M.T., and Parker, G.B., *Genetic Algorithms for the Development of Real-Time Multi-Heuristic Search Strategies*, in [129], pp.565–570, 1993.
[357] Shonkwiler, R. and Van Vleck, E., *Parallel Speed-up of Monte Carlo Methods for Global Optimization*, unpublished manuscript, 1991.
[358] Siedlecki, W. and Sklanski, J., *Constrained Genetic Optimization via Dynamic Reward–Penalty Balancing and Its Use in Pattern Recognition*, in [344], pp.141–150.
[359] Sirag, D.J. and Weisser, P.T., *Toward a Unified Thermodynamic Genetic Operator*, in [171], pp.116–122.
[360] Smith, A. and Tate, D., *Genetic Optimization Using A Penalty Function*, in [129], pp.499–503.
[361] Smith, D., *Bin Packing with Adaptive Search*, in [167], pp.202–207.
[362] Smith, R.E., *Adaptively Resizing Populations: An Algorithm and Analysis*, in [129], pp.653.
[363] Smith, S.F., *A Learning System Based on Genetic Algorithms*, PhD Dissertation, University of Pittsburgh, 1980.
[364] Smith, S.F., *Flexible Learning of Problem Solving Heuristics through Adaptive Search*, Proceedings of the Eighth International Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA, 1983.
[365] Spears, W.M., *Simple Subpopulation Schemes*, in [378], pp.296–307.
[366] Spears, W.M. and De Jong, K.A., *On the Virtues of Parametrized Uniform Crossover*, in [32], pp.230–236.
[367] Spears, W.M., *Adapting Crossover in Evolutionary Algorithms*, in [254], pp.367–384.
[368] Srinivas, M. and Patnaik, L.M., *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.24, No.4, 1994, pp.17–26.
[369] Srinivas, N. and Deb, K., *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*, Evolutionary Computation, Vol.2, No.3, 1994, pp.221–248.
[370] Starkweather, T., McDaniel, S., Mathias, K., Whitley, C., and Whitley, D., *A Comparison of Genetic Sequencing Operators*, in [32], pp.69–76.
[371] Stebbins, G.L., *Darwin to DNA, Molecules to Humanity*, W.H. Freeman, New York, 1982.
[372] Stein, D., *Scheduling Dial a Ride Transportation Systems: An Asymptotic Approach*, PhD Dissertation, Harvard University, 1977.
[373] Stevens, J., *A Genetic Algorithm for the Minimum Spanning Tree Problem*, MSc Thesis, University of North Carolina at Charlotte, 1991.
[374] Stuckman, B.E. and Easom, E.E., *A Comparison of Bayesian/Sampling Global Optimization Techniques*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.22, No.5, pp.1024–1032, 1992.



[375] Suh, J.-Y. and Gucht, Van D., *Incorporating Heuristic Information into Genetic Search*, in [171], pp.100–107.
[376] Surry, P.D., Radcliffe, N.J., and Boyd, I.D., *A Multi-objective Approach to Constrained Optimization of Gas Supply Networks*, presented at the AISB-95 Workshop on Evolutionary Computing, Sheffield, UK, April 3–4, 1995.
[377] Sverdlik, W. and Reynolds, R.G., *Incorporating Domain Specific Knowledge into Version Space Search*. Proceedings of the 1993 IEEE International Conference on Tools with AI, Boston, MA, November 1993, pp.216–223.
[378] Sebald, A.V. and Fogel, L.J., *Proceedings of the Third Annual Conference on Evolutionary Programming*, San Diego, CA, 1994, World Scientific.
[379] Steele, J.M., *Probabilistic Algorithm for the Directed Traveling Salesman Problem*, Mathematics of Operations Research, Vol.11, No.2, 1986, pp.343–350.
[380] Svirezhev, Yu.M., and Passekov, V.P., *Fundamentals of Mathematical Evolutionary Genetics*, Kluwer Academic Publishers, Mathematics and Its Applications (Soviet Series), Vol.22, London, 1989.
[381] Syslo, M.M., Deo, N., Kowalik, J.S., *Discrete Optimization Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
[382] Syswerda, G., *Uniform Crossover in Genetic Algorithms*, in [344], pp.2–9.
[383] Syswerda, G., *Schedule Optimization Using Genetic Algorithms*, in [78], pp.332–349.
[384] Syswerda, G. and Palmucci, J., *The Application of Genetic Algorithms to Resource Scheduling*, in [32], pp.502–508.
[385] Suh, J.-Y. and Lee C.-D., *Operator–Oriented Genetic Algorithm and Its Application to Sliding Block Puzzle Problem*, in [351], pp.98–103.
[386] Szalas, A., and Michalewicz, Z., *Contractive Mapping Genetic Algorithms and Their Convergence*, Department of Computer Science, University of North Carolina at Charlotte, Technical Report 006-1993.
[387] Taha, H.A., *Operations Research: An Introduction*, 4th ed., Collier Macmillan, London, 1987.
[388] Tamassia, R., Di Battista, G. and Batini, C., *Automatic Graph Drawing and Readability of Diagrams*, IEEE Transactions Systems, Man, and Cybernetics, Vol.18, No.1, pp.61–79, 1988.
[389] Ulder, N.L.J., Aarts, E.H.L., Bandelt, H.-J., van Laarhoven, P.J.M.,Pesch, E., *Genetic Local Search Algorithms for the Traveling Salesman Problem*, in [351], pp.109–116.
[390] Valenzuela, C.L. and Jones, A.J., *Evolutionary Divide and Conquer (I): A Novel Genetic Approach to the TSP*, Evolutionary Computation, Vol.1, No.4, 1994, pp.313–333.
[391] Vignaux, G.A. and Michalewicz, Z., *Genetic Algorithms for the Transportation Problem*, Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, North-Holland, Amsterdam, pp.252–259, 1989.
[392] Vignaux, G.A. and Michalewicz, Z., *A Genetic Algorithm for the Linear Transportation Problem*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.21, No.2, pp.445–452, 1991.
[393] Voss, S., *Tabu Search: Applications and Prospects*, Technical Report, Technische Hochshule Darmstadt, 1993.
[394] Whitley, D., *GENITOR: A Different Genetic Algorithm*, Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Denver, 1988.
[395] Whitley, D., *The GENITOR Algorithm and Selection Pressure: Why Rank–Based Allocation of Reproductive Trials is Best*, in [344], pp.116–121.
[396] Whitley, D., *GENITOR II: A Distributed Genetic Algorithm*, Journal of Experimental and Theoretical Artificial Intelligence, Vol.2, pp.189–214.
[397] Whitley, D., *Genetic Algorithms: A Tutorial*, in [263].
[398] Whitley, D. (Editor), *Foundations of Genetic Algorithms–2*, Second Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[399] Whitley, D., Gordon, V.S., and Mathias, K., *Lamarckian Evolution, the Baldwin Effect and Function Optimization*, in [70], pp.6–15.
[400] Whitley, D., Mathias, K., Fitzhorn, P., *Delta Coding: An Iterative Search Strategy for Genetic Algorithms*, in [32], pp.77–84.
[401] Whitley, D., Mathias, K., Rana, S., and Dzubera, J., *Building Better Test Functions*, in [103], pp.239–246.
[402] Whitley, D., Starkweather, T., and Fuquay, D'A., *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator*, in [344], pp.133–140.
[403] Whitley, D., Starkweather, T., and Shaner, D., *Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination*, in [78], pp.350–372.
[404] Wilson, R.B., *Some Theory and Methods of Mathematical Programming*, Ph.D. Dissertation, Harvard University Graduate School of Business Administration, 1963.
[405] Winston, W.L., *Operations Research: Applications and Algorithms*, Duxbury, Boston, 1987.
[406] Wirth, N., *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.
[407] Wnek, J., Sarma, J., Wahab, A.A., and Michalski, R.S., *Comparing Learning Paradigms via Diagrammatic Visualization*, Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems, North-Holland, Amsterdam, pp.428–437, 1990.
[408] Wright, A.H., *Genetic Algorithms for Real Parameter Optimization*, in [318], pp. 205–218.
[409] Xiao, J., *Evolutionary Planner/Navigator in a Mobile Robot Environment*, in [17], section G3.11.
[410] Xu, J. and Kelly, J.P., *A Robust Network Flow-Based Tabu Search Approach for the Vehicle Routing Problem*, Graduate School of Business, University of Colorado, Boulder, 1995.
[411] Yagiura, Y. and Ibaraki, T., *GA and Local Search Algorithms as Robust and Simple Optimization Tools*, Proceedings of the Metaheuristics International Conference, Breckenridge, Colorado, July 22–26, 1995, pp.129–134.
[412] Yao, X., *A Review of Evolutionary Artificial Neural Networks*, Technical Report, CSIRO, Highett, Victoria, Australia, 1993.
[413] Yao, X. and Darwen, P., *An Experimental Study of N-person Prisoner's Dilemma Games*, Proceedings of the Workshop on Evolutionary Computation, University of New England, November 21–22, 1994, pp.94–113.
[414] Zhou, H.H., and Grefenstette, J.J., *Learning by Analogy in Genetic Classifier Systems*, in [344], pp.291–297, 1989.
[415] Zurada, J., Marks, R., and Robinson, C. (Editors), *Computational Intelligence: Imitating Life*, IEEE Press, 1994.